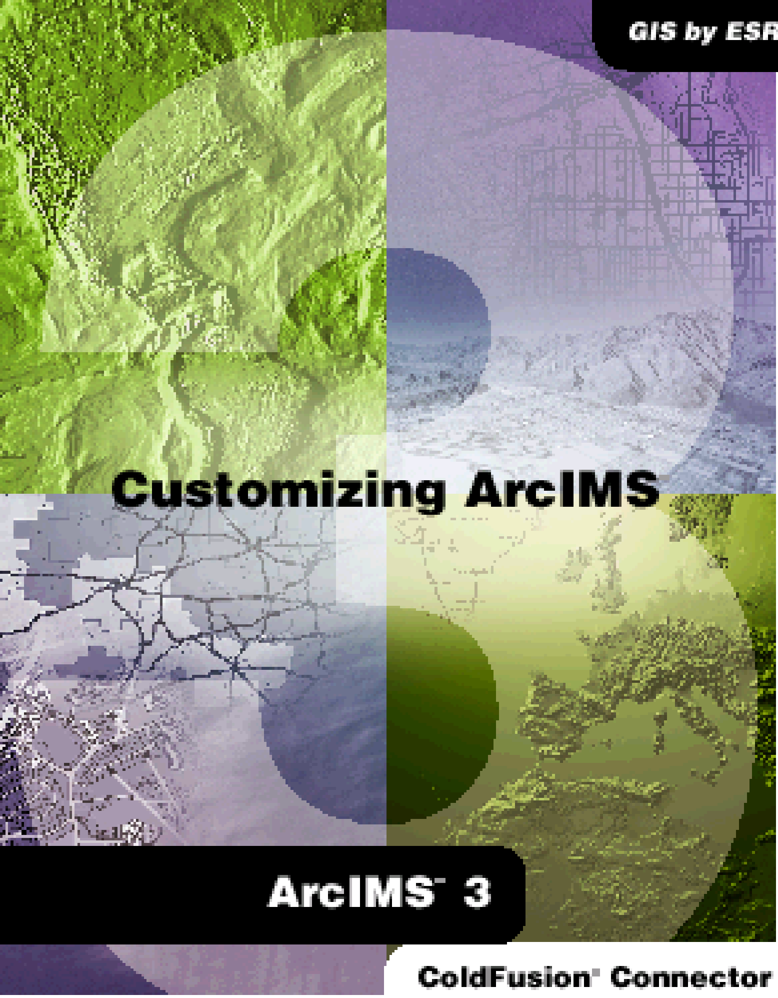


GIS by ESRI™

Customizing ArcIMS

ArcIMS™ 3

ColdFusion® Connector

ESRI

Contents

Introducing the ColdFusion Connector 1

IN THIS CHAPTER

ESRI® ArcIMS™ 3 software provides a suite of tools allowing you to create very effective web sites for your mapping and GIS needs. ArcIMS provides the foundation for the graphical and functional components of these web sites. You can build on this foundation through customization of ArcIMS.

Customizing ArcIMS is a series of programming reference books that describes the customization of the HTML and Java™ Viewers and the ActiveX® and ColdFusion® Connectors provided with ArcIMS.

This book explains the foundation for customizing a ColdFusion client, as well as provides a reference to the ArcXML tag specification designed to work with ColdFusion.

This book assumes that you have a working knowledge of HTML and building applications in ColdFusion. You should also have experience creating ArcIMS MapServices.

In this chapter, you are introduced to

- Reasons for customizing the ColdFusion Connector
- How different mapping functionality is implemented with ColdFusion
- The ColdFusion samples provided with ArcIMS

What is the ColdFusion Connector?

ArcIMS has three Application Server Connectors—standard, ActiveX and Coldfusion. In general, the job of a connector is to connect the web server to the ArcIMS Application Server. The ColdFusion Connector allows the ArcIMS Application Server to communicate to the ColdFusion server, which in turn communicates to the web server.

Considerations for choosing the ColdFusion Connector

ArcIMS provides four customizable clients—HTML, Java, ActiveX, and ColdFusion.

As a ColdFusion developer, you are already familiar with the power ColdFusion can afford you in working with databases over the Web. The ColdFusion Connector allows you to work with ArcIMS directly through a set of defined ColdFusion tags specific to GIS and mapping. The inclusion of these tags into a design-time control inside of ColdFusion Studio gives you the ability to put GIS mapping and database functionality onto your web sites as easily as you add any other tags.

You may want to consider these points when deciding to customize ArcIMS using ColdFusion:

- All requests are handled on the server-side offering three advantages—client-side processing is minimized, browser compatibility is reduced, and your client-side code is not exposed.
- Tags for performing sophisticated database programming tasks are simplified.
- The amount of programming expertise required is less than with conventional programming languages.
- A new ArcIMS toolbar is loaded into ColdFusion Studio to handle the new custom tags.
- Only Image Mapservices are supported because map images are created server-side.

ColdFusion sample applications setup

With the ColdFusion Connector, two sample viewer applications have been provided—the ColdFusion Template and the Sample Renderers. They demonstrate a variety of functions. A general set of instructions, along with a description and requirements for running each application, are provided below.

Samples included with ArcIMS

The ColdFusion Template is a general-purpose viewer that uses the ArcIMS custom ColdFusion tags to implement basic mapping functions. Functions like zoom, pan, identify, query, and geocoding have been incorporated in this template. The ColdFusion Template is a valuable starting point in creating custom applications. For example, you can use it as a simple ArcIMS data browser for Image MapServices then create a more sophisticated application later.

The Sample Renderer is not a general purpose viewer, but instead introduces some advanced ColdFusion Connector techniques. It demonstates using ArcXML requests to modify the rendering on a MapService.

Installing the samples

Before you can use the sample applications, you must have the ColdFusion Server, ColdFusion Studio, and the ArcIMS ColdFusion Connector installed and configured. The instructions on how to install and configure the ColdFusion Connector are in the *ArcIMS 3.0 Installation Guide*, 'Installing ArcIMS on Windows NT: ArcIMS custom Application Server Connectors', and Appendix C, 'Configuring ColdFusion for the ColdFusion Connector'.

To install the samples, start a Custom Installation of ArcIMS. During the installation, check Samples from the ArcIMS components dialog and click Options. Check Sample ColdFusion Applications and click OK. This installs the samples to <ArcIMS installation directory>\Samples\ColdFusion. Two directories are included:

<ArcIMS installation directory>\Samples\ColdFusion\ColdFusionTemplate—contains the CFM files that make up the template. Also included is an \images directory that contains the GIF files used as icons for the buttons on the template.

<ArcIMS installation directory>\Samples\ColdFusion\SampleRenderers—contains CFM files and images directory that make up the rendering application.

Before you can view the samples, you must do one of the following:

- Copy the \ColdFusion directory to your \ArcIMS\ Website directory.
- Copy the \ColdFusion directory to your web server root directory.
- Create virtual directories to the current location of \ColdFusion.

Using the ColdFusion Template

The ColdFusion Template is a framework for using the ColdFusion Connector. The template shows the foundation of an application built with ColdFusion. Functions like zoom, pan, identify, query, add layers, and change symbols have been incorporated in this template. The ColdFusion Template is a valuable starting point in creating custom applications.

Starting the SanFrancisco MapService

The SanFrancisco Image MapService used in the template is based on sf.axl. Sf.axl can be found in the <ArcIMS installation directory>\Samples\Viewers\AXL subdirectory.

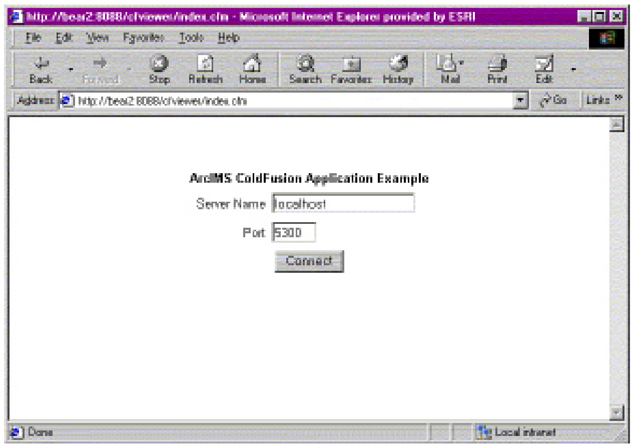
The sf.axl file uses data from the C:\Program Files\ESRI\ArcIMS3.0\Samples\TutorialData, the default installation folder. If you did not accept the default installation location, you must change the sf.axl file to look for the sample data in the location you specified. The directory attribute of the SHAPEWORKSPACE tag must be changed as follows:

```
<SHAPEWORKSPACE name=”shp_ws-60"
directory=”<My Installation Location>\Samples\TutorialData” />
```

In ArcIMS Administrator, use the sf.axl file to create an Image MapService named SanFrancisco (case sensitive). See *Using ArcIMS* for instructions on creating MapServices.

Opening the template

In a browser, type http://<local host>/website/ColdFusion/ColdFusionTemplate/index.cfm to open the template. A form with the Server Name and Port input fields appears.


http://bear2:8088/cfviewer/index.cfm - Microsoft Internet Explorer provided by ESRI
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://bear2:8088/cfviewer/index.cfm
Go Links
Done
Local intranet

ArcIMS ColdFusion Application Example

Server Name localhost

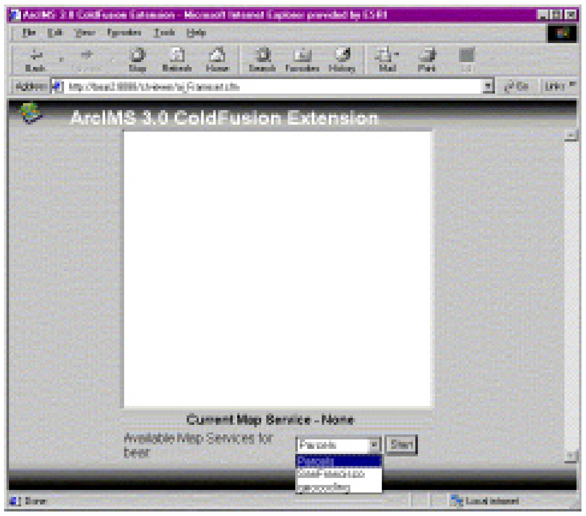
Port 5300

Connect

Type in the Server name and port number where the ArcIMS Application Server is running then click Connect.

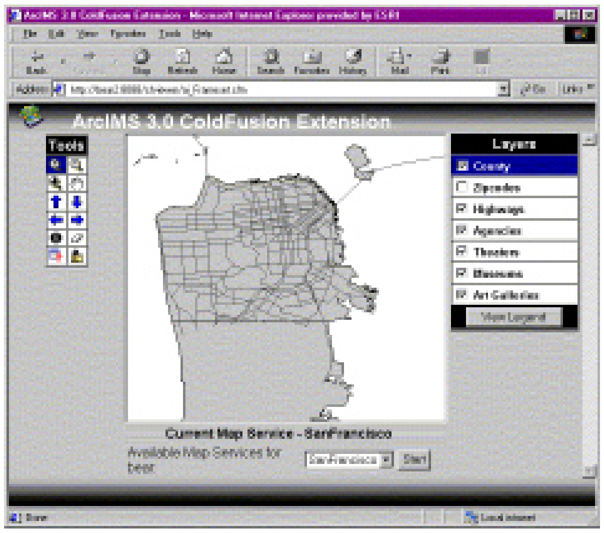
The main page appears with an area for the map and a scrolling list of MapServices.

Note: If you don't get a list of MapServices on the scrolling list and receive an error about CFX_ESRIMAP, you haven't correctly configured ColdFusion for ArcIMS. Follow the configuration steps outlined in the *ArcIMS 3.0 Installation Guide,* Appendix C, 'Configuring ColdFusion for the ColdFusion Connector', and try connecting to the sample again.


ArcIMS 3.0 ColdFusion Extension
Current Map Service - None
Available Map Services for bear
Parcels
Start

Select the SanFrancisco MapService and click Start.

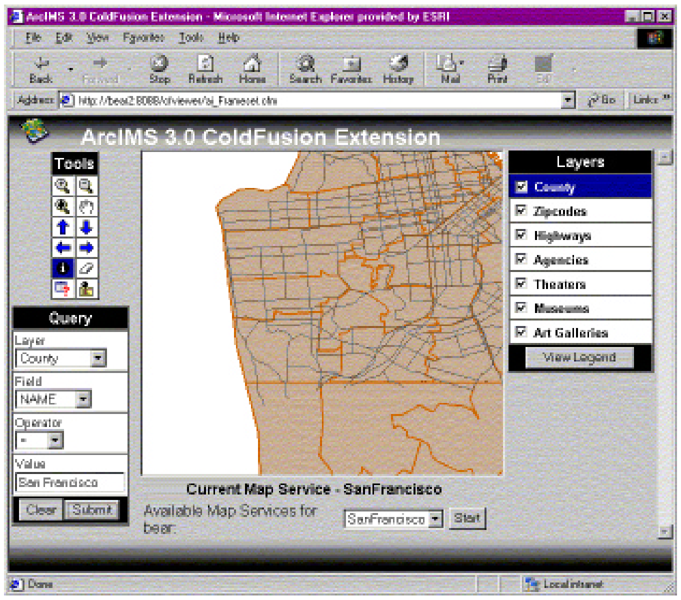
A list of layers should appear on the right, a toolbar on the left, and a map image in the middle.



Layer list and toolbar interface

The template shows how to work with the layer list and toolbar functions of the interface.

- Layers can be turned on or off using check-boxes in the layer list. The point layers such as Art Galleries, have scale factors set, so they don't display until you zoom in.
- The View Legend button, located below the layer list, opens the legend in a new browser window. There is a known issue with point layers not appearing on the legend image.
- Each tool displays its name via a tooltip when you hover the cursor over the tool for a couple of seconds.
- The ZoomIn tool allows the user to center and zoom into the map on the point clicked on the map image. This viewer does not support zoom to rectangle
- The ZoomOut tool works the same way—allows the user to center and zoom out from the point clicked on the map image.
- The Pan tool allows the user to reposition the map centered on the point clicked on the map.
- Pan North, Pan South, Pan East and Pan West buttons pan North, South, East and West respectively approximately half of the map extent.
- Identify tool identifies selected features from the active layer. Select a layer in the legend to make it active, click on the tool, and then click a feature on the map image. A new window appears with the attributes of the identified feature.
- Query Form button opens a new form below the toolbar. This form is used to perform an attribute query on a layer. To make an attribute query, select a layer, select a field, select an operator, and type in the value you are looking for. For example, choose County, and create the query Name = San Francisco. A table of the selected features appears in a new browser window.

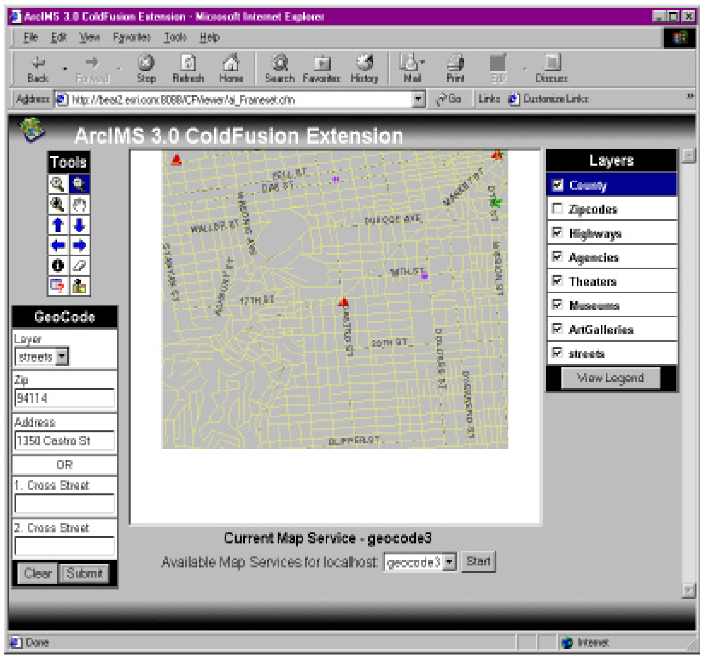

ArcIMS 3.0 ColdFusion Extension - Microsoft Internet Explorer provided by ESRI
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address http://beai2:8088/cfviewer/ai_Frameset.cfm
Go Links
ArcIMS 3.0 ColdFusion Extension
Tools
Query
Layers
County
Zipcodes
Highways
Agencies
Theaters
Museums
Art Galleries
View Legend
Clear
Submit
Start
Done
Local intranet

Layer: County
Field: NAME
Operator: =
Value: San Francisco

Current Map Service - SanFrancisco
Available Map Services for bear: SanFrancisco

- The GeoCode Form tool displays a form to type in your address for geocoding. This tool requires a MapService with geocoding properties set up. To create this MapService, you can either update the SanFrancisco MapService or by create a new MapService. Whichever you choose, add in the layer C:\Program Files\ESRI\ArcIMS3.0\Samples\TutorialData\streets, and set the geocodeing properties to US Street with Zone. To get more information how to prepare a MapService for geocoding, see *Using ArcIMS*.

  Address can be geocoded based on a street address or on a street intersection. To get the address point location, select the layer from the Layer list, and fill ZIP and Address fields, or fill Cross Street fields, and click Submit. A list of candidates, their coordinates, and their scores is opened in a new browser window.


ArcIMS 3.0 ColdFusion Extension - Microsoft Internet Explorer
ArcIMS 3.0 ColdFusion Extension
Tools
GeoCode
Layers
View Legend
Clear
Submit
Start

Layer: streets
Zip: 94114
Address: 1350 Castro St
OR
1. Cross Street:
2. Cross Street:

Layers:
✓ County
☐ Zipcodes
✓ Highways
✓ Agencies
✓ Theaters
✓ Museums
✓ ArtGalleries
✓ streets

Current Map Service - geocode3
Available Map Services for localhost: geocode3

ColdFusion Template files

The following files are included in the ColdFusion Template. All files are located in the <ArcIMS installation directory>\Samples\ColdFusion\ColdFusionTemplate. See 'Installing the samples' earlier in the chapter if you can't locate the directory. They are briefly introduced here and are discussed in more detail later in this chapter.

- ai_Blank.cfm—displays blank silver background for ai_QueryForm.cfm.
- ai_Bottom.cfm—displays the gray graphic at the bottom of the template.
- ai_Frameset.cfm—creates the frame structure of the page and gathers server access information.
- ai_GeoCode.cfm—processes geocode request from ai_GeoCodeForm.cfm and displays the results in HTML table.
- ai_GeoCodeForm.cfm—contains user interface for building geocode requests and sends form to ai_GeoCode.cfm.
- ai_Globals.cfm—contains the CFApplication declaration; sets attributes for server access.
- ai_Identify.cfm—displays the results of the Identify function.
- ai_Layers.cfm—turns layers on/off and selects the active layer. Uses ui.js for toggling layers.
- ai_Legend.cfm—shows the legend image.
- ai_Map.cfm—generates a new map.
- ai_Query.cfm—processes query from ai_Query.cfm and displays the results in HTML table.
- ai_QueryForm.cfm—contains user interface for building queries and sends form to ai_Query.cfm.
- ai_Results.cfm—not used in template application.
- ai_Service.cfm—allows users to select a MapService. This is the first screen of the template.
- ai_Toolbar.cfm—defines structure and contains icons for toolbar. Uses ui.js for toggling functionality.
- ai_Top.cfm—displays the gray header graphic and ArcIMS logo at the top of the template.
- index.cfm—starts the template application.
- ui.js—JavaScript file containing a function to handle the dynamic toolbar.

ColdFusion Template frames

The ColdFusion Template is composed of a series of files that open in sequence.The starting page, index.cfm, initializes the application then calls ai_Frameset.cfm which divides the main page of the template into seven frames—TopFrame, ToolFrame, QueryFrame, MapFrame, ServicesFrame, LayersFrame, and BottomFrame.

TopFrame and BottomFrame

The TopFrame and BottomFrame hold the graphics for the top and bottom of the template. Ai_Top.cfm and ai_Bottom.cfm fill theses frames.

ToolFrame

The ToolFrame holds the toolbar for the template. The file ai_Toolbar.cfm fills the ToolFrame.

For many of the tools, when one is clicked it is selected but not executed. Instead, these tools are executed when the Map is clicked or a Submit button is clicked. The JavaScript function switchTool, in ui.js, gets called when a tool is clicked.

QueryFrame

The QueryFrame is positioned below the toolbar. It initially holds ai_Blank.cfm then is replaced by either the query or geocoding interface found in ai_QueryForm.cfm or ai_GeoCodeForm.cfm, respectively.

MapFrame

The MapFrame holds the map for the template. The purpose of the MapFrame is to hold the map image and the hidden inputs passed from the ToolFrame. The file ai_Map.cfm opens the MapFrame.

ServicesFrame

The ServicesFrame is positioned below the map frame and provides the user with a list of MapServices. ai_Services.cfm fills the ServicesFrame.

LayersFrame

The LayersFrame holds the layer list for a MapService. The file ai_Layers.cfm fills the LayerFrame. The layer list is dynamically generated when a MapService is chosen. The LayerFrame controls three types of functionality: controlling layer visibility, setting the active layer, and toggling between the layer list and the legend.

Functionality of the ColdFusion Template

In this section you are introduced to the ColdFusion techniques used to implement the functionality of the ColdFusion Template presented earlier in the chapter. Each section builds on the previous to construct the template application.

Setting server access

The Coldfusion Template provides a file named ai_Globals.cfm. The purpose of this file is to set global variables for the session that define server access. For example, ai_Globals.cfm sets the ServerName and ServerPort that describe access to the server. These attributes are the basis for having a successful connection to the server.

```
<!--
   ai_Globals.cfm
   CFApplication declaration; sets default attributes for server access; sets
   global attributes
->
<cfapplication name=”CF ARCIMS Sample” sessionManagement=”yes”>
<cfparam name=”Session.ServerName” default=”localhost”>
<cfparam name=”Session.ServerPort” default=”5300">
<cfif NOT isDefined(“Session.Connected”)>
    <!- Session expired or never existed, throw user back to index.cfm ->
   <cflocation url=”index.cfm”>
</cfif>
```

Opening page and frames

The starting page, index.cfm, contains a simple form to get the host name and port and calls ai_frameset.cfm. Ai_Frameset.cfm reads in the server information from ai_Globals.cfm and divides the main page of the template into seven frames—TopFrame, ToolFrame, QueryFrame, MapFrame, ServicesFrame, LayersFrame, and BottomFrame. The map frame is in the middle, the tool frame is on the left, the legend is on the right, and the MapServices frame is on the bottom. When the page first opens, only the bottom frame, ServicesFrame contains content.

Getting the MapServices

The source file for ServicesFrame is ai_Service.cfm. This file contains the CF_ARCIMS GetMapServices request, composed of the <CF_ARCIMS> tag and action “GetMapServices”, and creates a form with a drop-down list of MapServices returned in the response.

```
<CF_ArcIMS Action="GetMapServices"
   ServerName="#Session.ServerName#"
   ServerPort="#Session.ServerPort#">
<!– If nothing is returned, then give user error message. –>
<cfif not isDefined("Out_ServicesTable")>
   <cflocation url="index.cfm?error=#Out_Error#&ServerName=#Session.ServerName#&
   ServerPort=#Session.ServerPort#">
</cfif>
```

Notice the technique used above checks if any MapServices are returned. If the host has no MapServices, Out_ServicesTable is not created. The programmer therefore, must check for the existence of the table before trying to use any values in it.

Here is the code in ai_Service.cfm that creates the form containing the drop-down list of MapServices:

```
<form id=SelectService method="post" action="ai_Frameset.cfm" target="_top">
<input type=hidden name="ServerName" value=#Session.ServerName#>
<input type=hidden name="ServerPort" value=#Session.ServerPort#>
<tr>
<td>
   <font face="arial" size=3>
      Available MapServices for #Session.ServerName#:
   </font>
</td>
<td>
   <!– Build available mapservice selection list from the info returned by
getservices action –>
   <select name="ServiceName
   <cfloop Query="OUT_ServicesTable">
      <cfloop index="curName" list="#Out_ServicesTable.Name#">
        <option value=#curName# <cfif #curName# eq #Session.ServiceName#>SELECTED
        </cfif>>#curName#
      </cfloop>
   </cfloop>
   </select>
</td>
<td>
   <input type=submit value="Start">
</td>
</tr></table></form>
```

The <cfloop Query> loop goes through the Out_ServicesTable to create <option> tags for the <select> field. Notice the <cfif> tag is used to check if there is already a MapService selected in the Session.ServiceName variable, and selects it by default.

Selecting a MapService from the list

After a MapService is selected, ai_Frameset.cfm gets reloaded. The section of code below contains the CF_ARCIMS GetServiceInfo request. All of the layer and field information is stored in the Session.LayerInfo structure. Notice how structure is used to store entire ColdFusion query.

```
<CF_ArcIMS Action=”GetServiceInfo” ServerName=”#Form.ServerName#”
ServerPort=”#Form.ServerPort#” ServiceName=”#Form.ServiceName#”>
<cfif isDefined(“OUT_LayerTable”)>
   <!— Store server info in Session variables —>
   <cfset Session.ServerName = Form.ServerName>
   <cfset Session.ServerPort = Form.ServerPort>
   <cfset Session.ServiceName = Form.ServiceName>
   <cfset Session.Initialize = “true”>

   <!— Store layer information in a Session Struct variable —>
   <cfset Session.LayerInfo = StructNew()>
   <cfset temp = StructInsert(Session.LayerInfo, “LayerTable”, OUT_LayerTable)>
   <cfset temp = StructInsert(Session.LayerInfo, “ActiveLayerID”,
   OUT_LayerTable.ID)>
   <cfset temp = structinsert(session.LayerInfo, “ActiveLayerName”,
   OUT_LayerTable.Name)>

   <!— Set up list of layer names that have field tables —>
   <cfset LayerFieldList = “”>
   <cfloop query=”Out_LayerTable”>
      <cfif Featuretype neq “Image”>
        <cfset LayerFieldList = Listappend(LayerFieldList, Name)>
      </cfif>
      </cfloop>

   <!— Loop through Layers and set field query objects to LayerInfo struct —>
   <cfloop index=”curLayer” list=”#LayerFieldList#”>
        <cfset temp = StructInsert(Session.LayerInfo,“#curLayer#_FIELDTABLE”,
        Evaluate(“OUT_#curLayer#_FIELDTABLE”))>
   </cfloop>
```

```
    <!— Store extent information in a Session Struct variable —>
    <cfset temp = StructInsert(Session.LayerInfo,“MaxX”,OUT_MapEnvelopeTable.MaxX)>
    <cfset temp = StructInsert(Session.LayerInfo,“MaxY”,OUT_MapEnvelopeTable.MaxY)>
    <cfset temp = StructInsert(Session.LayerInfo,“MinX”,OUT_MapEnvelopeTable.MinX)>
    <cfset temp = StructInsert(Session.LayerInfo,“MinY”,OUT_MapEnvelopeTable.MinY)>
    <cfset temp = StructInsert(Session.LayerInfo,“MMaxX”,OUT_MapEnvelopeTable.MaxX)>
    <cfset temp = StructInsert(Session.LayerInfo,“MMaxY”,OUT_MapEnvelopeTable.MaxY)>
    <cfset temp = StructInsert(Session.LayerInfo,“MMinX”,OUT_MapEnvelopeTable.MinX)>
    <cfset temp = StructInsert(Session.LayerInfo,“MMinY”,OUT_MapEnvelopeTable.MinY)>

    <!— Store default image dimensions —>
    <cfset temp = StructInsert(Session.LayerInfo, “ImageWidth”, “400”)>
    <cfset temp = StructInsert(Session.LayerInfo, “ImageHeight”, “300”)>
    <cfelse>
      <!— We can’t connect to the server, reload index.cfm with an error message —>
      <cflocation url=”index.cfm?error=#OUT_Error#&ServerName=#Form.ServerName#&
      ServerPort=#Form.ServerPort#”>
</cfif>
```

Creating the layer list and legend

Ai_Frameset.cfm calls ai_Layers.cfm. This section of code from ai_Layers.cfm uses the Out_LayerTable stored in the Session.LayerInfo structure to create a list of layers.

```
<form name=”LayerForm” action=”ai_Map.cfm” target=”MapFrame” method=”post”>
<input type=”hidden” name=”layers” value=””>
<table style=”cursor:hand;” border=”1" cellspacing=”0" cellpadding=”2"
align=”CENTER” valign=”MIDDLE” bgcolor=”white” bordercolor=”Gray”
bordercolorlight=”Silver” bordercolordark=”Black” width=”95%”>
<tr bgcolor=”000000">
    <td align=”center” title=”Layer Control Form”><b><font face=”Arial” size=3
    color=”ffffff”>Layers</font></b></td>
</tr>
```

(continued on next page)

```
<!— Loop through the layers, to build layer menu —>
<cfoutput query="Session.LayerInfo.LayerTable">
<tr>
   <td id="#name#" onMouseDown="switchTool(this, 'yes');
      parent.MapFrame.MapForm.ActiveLayerID.value ='#id#';
      parent.MapFrame.MapForm.ActiveLayerName.value = '#Name#'">
      <input type="checkbox" name="layers" value="#name#"
      onClick="LayerForm.submit();"
      <cfif visible eq "true">checked</cfif>>
      <b><font face="Arial" size=2>
      #replacelist(name, "_ESRIDOT_,_ESRIDASH_", ".,-")# </font></b>
   </td>
</tr>
</cfoutput>
<tr bgcolor="000000">
   <td align="center"><input type="button" value="View Legend" onClick="NewWin()";
   </td>
</tr>
</table>
```

The checkbox on the onClick event handler is associated with Form.Submit. This ensures the map is refreshed every time user turns a layer on or off. Also, ' _ESRIDOT_' and '_ESRIDASH_' are used to replace with dots and dashs to recreate original layer names. Dots and dashs are not allowed in ColdFusion variable names, but are legal in ArcIMS layer names.

The 'View Legend' button has an associated JavaScript function, named NewWin, that loads ai_Legend.cfm into a new browser window.

The form action reloads ai_Map.cfm to the map frame.

Using the map interaction tools—zoom, pan, and identify

Almost every map application requires some kind of user interaction with the map. In the ColdFusion Template, most of the tool interaction is handled in ai_Map.cfm. It is reloaded from itself each time the user clicks on the map to do zoom, pan, or identify.

Although the user clicks the toolbar to choose the function he wants to perform, ai_Toolbar.cfm only contains the interface elements and defines the onMouseDown event. The onMouseDown event calls the JavaScript function named switchTool from ui.js.

The following code is excerpted from ai_Map.cfm. Each time a tool used on the map this code gathers information about the click location and layers, then creates a new image. The action on the form is a ColdFusion system variable that points to the current cfm file.

The OUT_ImageURL variable is used in the <input> tag. The <input> type image is used for the map, instead of the <img> tag, because it is the easiest way to capture click coordinates. Click coordinates are captured in the image coordinate system and are calculated into a map coordinate system.

```
<!– Setup form to use when user clicks on map –>
<form name=”MapForm” action=”#cgi.script_name#” method=”post”>
<input type=”hidden” name=”ImageWidth” value=”#Session.LayerInfo.ImageWidth#”>
<input type=”hidden” name=”ImageHeight” value=”#Session.LayerInfo.ImageHeight#”>
<input type=”hidden” name=”MinX” value=”#OUT_MinX#”>
<input type=”hidden” name=”MinY” value=”#OUT_MinY#”>
<input type=”hidden” name=”MaxX” value=”#OUT_MaxX#”>
<input type=”hidden” name=”MaxY” value=”#OUT_MaxY#”>
<input type=”hidden” name=”ActiveLayerID”
   value=”#Session.LayerInfo.ActiveLayerID#”>
<input type=”hidden” name=”ActiveLayerName”
   value=”#Session.LayerInfo.ActiveLayerName#”>
<input type=”hidden” name=”SelectionLayer” value=”#Session.SelectionLayer#”>
<input type=”hidden” name=”SelectionField” value=”#Session.SelectionField#”>
<input type=”hidden” name=”SelectionFieldValuesList”
   value=”#Session.SelectionFieldValuesList#”>
<input type=”hidden” name=”Tool” value=”#Session.Tool#”>
<cfif isDefined(“Form.FieldNames”) or (CGI.HTTP_USER_AGENT does not contain
“MSIE”)>
   <div align=”center”>
   <input name=”Map” type=”image” src=”#OUT_ImageURL#”
   width=”#Session.LayerInfo.ImageWidth#” height=”#Session.LayerInfo.ImageHeight#”
   border=0>
   </div>
</form>
```

How the ZoomIn, ZoomOut, and Pan tools work

The following code is excerpted from ai_Map.cfm. The Map.X and Map.Y variables are defined with the X and Y coordinates of the click. In this case, a new map extent is calculated. There is also a section of code in the file for directional pan and zoom to full extent.

```
<!– Calculate the x y from image width –>
<cfif isDefined(“Map.X”)>
   <cfset CurWidth = MaxX - MinX>
   <cfset CurHeight = MaxY - MinY>
   <cfset X = MinX + CurWidth * (Map.X / ImageWidth)>
   <cfset Y = MaxY - CurHeight * (Map.Y / ImageHeight)>
   <cfswitch expression=”#Session.Tool#”>
      <!– If user clicked on map with zoom or pan tools, calculate new LRTB –>
      <cfcase value=”ZoomIn,ZoomOut,Pan”>
        <cfset ZoomFac = IIF(ListFindNoCase(Session.Tool, “ZoomIn”), ZoomFactor,
        “1/#ZoomFactor#”)>
        <cfset ZoomFac = IIF(ListFindNoCase(Session.Tool, “Pan”), 1, ZoomFac)>
        <cfset NewWidth = CurWidth / ZoomFac>
        <cfset NewHeight = CurHeight / ZoomFac>
        <cfset Session.LayerInfo.MaxX = X + (NewWidth / 2)>
        <cfset Session.LayerInfo.MaxY = Y + (NewHeight / 2)>
        <cfset Session.LayerInfo.MinX = X - (NewWidth / 2)>
        <cfset Session.LayerInfo.MinY = Y - (NewHeight / 2)>
      </cfcase>
```

Since the ColdFusion ArcIMS Connector does maintain state, it is necessary to keep map extent information in session variables, and recalculate it for every zoom or pan. Also, there is no Zoom In or Zoom Out functions provided by the connectors, therefore the application must calculate a new map extent and give it to every GenerateMap request.

A GenerateMap request follows.

```
<CF_ArcIMS action=”GenerateMap”
   ServerName=”#Session.ServerName#”
   ServerPort=”#Session.ServerPort#”
   ServiceName=”#Session.ServiceName#”
   Envelope=”#Session.LayerInfo.minx#,#Session.LayerInfo.miny#,
   #Session.LayerInfo.maxx#,#Session.LayerInfo.maxy#”
   Imagesize=”#Session.LayerInfo.ImageWidth#,#Session.LayerInfo.ImageHeight#”
   TransparencyColor=”255,255,0"
   LayerListOrder=”false”
   SelectionLayer=”#session.SelectionLayer#”
   SelectionField=”#session.SelectionField#”
   SelectionFieldValuesList=”#session.SelectionFieldValuesList#”
   SelectionFillColor=”255,255,0"
   SelectionBoundaryColor=”0,0,0">
   <!– Pass layer values to sub tag for processing –>
   <cfloop query=”Session.LayerInfo.LayerTable”>
      <CF_ArcIMS_Layer layerID=”#id#” visible=”#visible#”>
   </cfloop>
</CF_ArcIMS>
```

Most of the map parameters are stored in session variables. This is typical for an ArcIMS ColdFusion application. Layer visibility is set by looping through the LayerTables stored in the Session.LayerInfo structure and using the CF_ARCIMS_LAYER tag to get the visibility.

How the Identify tool works

The Identify tool works in a very similar way. All that is required are the X and Y coordinates of the user’s click on the map. Ai_Map.cfm launches ai_Identify.cfm, and the coordinates are contained in the variables PosX and PosY.

Below is the code excerpted from ai_Map.cfm that displays the results of the Identify.

```
<script>
   QueryWin = window.open(“ai_Identify.cfm?x=#Map.X#&y=#Map.Y#&
   layer=#activelayerID#&LayerName=#Activelayername#”,
   “QueryResults”,“toolbar=no,location=no,directories=no,status=no,resizable=yes,
   scrollbars=yes,width=400,height=300”);
</script>
```

The following code is from ai_Identify.cfm.

```
<!– Identify action –>
<CF_ArcIMS action=”Identify”
   ServerName=”#Session.ServerName#”
   ServerPort=”#Session.ServerPort#”
   ServiceName=”#Session.ServiceName#”
   LayerID=”#Layer#”
   PosX=”#X#”
   PosY=”#Y#”
   Envelope=”#Session.LayerInfo.minx#,#Session.LayerInfo.miny#,
   #Session.LayerInfo.maxx#,#Session.LayerInfo.maxy#”
   Imagesize=”#Session.LayerInfo.ImageWidth#,#Session.LayerInfo.ImageHeight#”
   Tolerance=1>
```

#X# and #Y# are the same as #Map.X# and #Map.Y#. The Identify request creates the map envelope and performs the image coordinate to map coordinate transformation.

The following code from ai_Identify.cfm creates the output table columns, minus the shape, ID, and x,y fields.

```
<cfif isdefined(“OUT_QueryTable”)>
<table bgcolor=”000000" cellpadding=3 cellspacing=1 border=1>
<tr>
   <!– Loop through the output table to grab column names –>
   <cfloop index=”curCol” list=”#Out_QueryTable.ColumnList#”>
      <!– Grab the ID field to use as the selection hightlight field –>
      <cfif curCol eq “POUND_ID_POUND”>
        <cfset FieldName=”ID”>
      </cfif>
      <!– Do not display the ID or Shape fields in the results –>
      <cfif (not curCol eq “pound_shape_pound”) and (not curCol eq
      “pound_id_pound”) and (not curCol eq “minx”) and (not curCol eq “miny”)
      and (not curCol eq “maxx”) and (not curCol eq “maxy”)>
      <cfoutput>
      <td>
        <font face=”Arial” size=”2" color=”ffffff”><b>#curCol#</b></font>
      </td>
      </cfoutput>
      </cfif>
   </cfloop>
</tr>
```

Using the Query tool

Ai_QueryForm.cfm creates the query form for the user input. It is positioned below the toolbar and uses Session.Layerinfo structure to get the names of the fields. After the user submits the query, ai_Query.cfm is loaded into a new browser window with the results. The CF_ArcIMS query request is executed with the parameters from the form and results are formatted to a table. The following is excerpted from ai_Query.cfm.

```
<!– Sql Query Action –>
<CF_ArcIMS action=”Query”
   ServerName=”#Session.ServerName#”
   ServerPort=”#Session.ServerPort#”
   ServiceName=”#Session.ServiceName#”
   ReturnEnvelope=”false”
   LayerID=”#Form.LayerID#”
   Where=”#Form.QueryField# #Form.WhereClause# #SQLClause#”>

<body bgcolor=”silver” text=”000000">
<font face=”arial” size=2>
<cfoutput>
   <u>QUERY RESULTS</u>: <b>#Ucase(replacelist(layername, “_ESRIDOT_,_ESRIDASH_”,
   “.,-”))# - WHERE #form.QueryField# #form.WhereClause# #form.SQLClause#</b><br>

   <!– Determine if any results were returned –>
   <cfif not isdefined(“OUT_QueryTable”)>
      <br>
      <div align=”center”>No features found.</div>
   <cfelse>
      Records: <b>#OUT_QueryTable.RecordCount#</b><br>
   </cfif>
</cfoutput>
```

As shown above, it is good practice to check the existence of the OUT_QueryTable with <cfif> and to report the number of selected records in the results window.

The code continues to define the output table columns and values. Some field name checking is done and results are printed to the HTML table. Field name checking is done to elimate fields of no interest to the user. In this case fields containing the internal ID, Shape field (as integer), and AXL_GEOMETRY field containing the geometry of the feature are not displayed in the table. The code for defining the columns of the output table follows.

```
<!— If a result is returned, then build a table to display results —>
<cfif isdefined("OUT_QueryTable")>
<table bgcolor="000000" cellpadding=3 cellspacing=1 border=1>
<tr>
   <!— Loop through results to grab column names —>
   <cfloop index="curCol" list="#Out_QueryTable.ColumnList#">
      <!— Set the selection field highlight to the ID field —>
      <cfif curCol eq "POUND_ID_POUND">
        <cfset FieldName = "ID">
      </cfif>
      <!— Do not display the shape or id fields —>
      <cfif (not curCol eq "pound_shape_pound") and (not curCol eq
      "pound_id_pound")>
      <cfoutput>
      <th>
        <font face="Arial" size="2" color="ffffff"><b>#curCol#</b></font>
      </th>
      </cfoutput>
      </cfif>
   </cfloop>
</tr>
```

Using the Geocoding tool

When you choose the Geocoding tool, ai_GeoCodeForm.cfm loads and creates the geocoding form for the user input. The Geocode tool requires that your MapService has a layer with defined Geocoding properties and the user input form only supports US Street or US Street with Zone geocoding styles. If you are not sure what geocoding style is associated with a layer, check the Out_GeoCodeStyles table, or look at the Map configuration file (.axl). See 'Using the ColdFusion Template' earlier in the chapter for more information on setting up your MapService to work with the Geocode tool.

Ai_GeocodeForm.cfm makes a CF_ArcIMS GetServiceInfo request to establish session information. The Out_GeoCodeStyles table contains names of the layers that can be used for the geocoding. If no layers in the selected MapService can be used for geocoding, an Out_GeoCodeStyles table is not created.

The following code is taken from ai_GeoCodeForm.cfm.

```
<!— Send GetServiceInfo request to find geocode defined layers —>
<cf_ArcIMS Action="Getserviceinfo"
   Servicename="#session.ServiceName#"
   ServerName="#Session.ServerName#"
   ServerPort="#Session.ServerPort#">

<body bgcolor="silver">

<!— Do not show geocode panel, if no map service has been choosen —>
<cfif "#Session.ServiceName#" neq "None">

<!— Form used to get user input to send to server —>
<form action="ai_GeoCode.cfm" method="post" target="GeoCodeResults"
onSubmit="NewWin();">
<table  border="1" cellspacing="0" cellpadding="2" bgcolor="white"
bordercolor="Gray" bordercolorlight="Silver" bordercolordark="Black">
   <tr bgcolor="000000">
      <td align=center title="Dynamic GeoCode Form"><b><font face="Arial" size=3
      color="ffffff">GeoCode</font></b></td>
   </tr>
<!— Check to see if geocodestyles was returned —>
<cfif parameterexists(out_geocodestyles)>
<cfif not isdefined("Layer")>
   <cfset Layer = "#out_geocodestyles.layername#">
</cfif>
   <tr>
      <td>
      <!— Select list for layer to geocode on. When user changes selection, script
      updates value in "fields" select list —>
      <font face="Arial" size=2>Layer</font><br><select name="LayerID"
      onchange="location = 'ai_GeoCodeForm.cfm?Layer=' +
      LayerID.options[LayerID.selectedIndex].id">
      <cfoutput>
```
(continued on the next page)

```
        <!— Loop through geocodestyles query object to build layer choices —>
        <cfloop query=”Out_GeoCodeStyles”>
              <cfloop query=”Out_LayerTable”>
                    <cfif #Out_GeoCodeStyles.LayerName# eq #Out_LayerTable.Name#>
                          <cfset LayerID = #Out_LayerTable.ID#>
                    </cfif>
              </cfloop>
              <option  value=”#LayerID#” id=”#LayerName#” <cfif #LayerName# eq
              #Layer#>SELECTED</cfif>>#LayerName#
        </cfloop>
      </cfoutput>
      </select>
   </td>
```

Most of the other procedures require the layer ID, contained in Out_LayerTable, but Out_GeoCodeStyles returns the layer name. The code loops through Out_LayerTable and the Out_GeocodeStyles table, compares every layer name, and returns the layer ID.

When the geocoding form is submitted, ai_GeoCode.cfm gets loaded into a new browser window. It sends the geocode request, checks the existance of OUT_GeocodeResults, builds and presents the result. See the section of code below.

```
<!— Geocode action —>
<CF_ArcIMS action=”GeoCode”
   ServerName=”#Session.ServerName#”
   ServerPort=”#Session.ServerPort#”
   ServiceName=”#Session.ServiceName#”
   LayerID=”#form.LayerID#”
   Street=”#Street#”
   CrossStreet=”#CrossStreet#”
   Zone=”#form.Zip#”
   PinPoint=”true”>


<body bgcolor=”silver” text=”000000">
<font face=”arial” size=2>
<cfoutput>
<u>GEOCODE RESULTS</u>: <b>#Ucase(form.LayerName)# - #Street# <cfif CrossStreet eq
“”>#Zip#<cfelse>and #CrossStreet# #Zip#</cfif></b><br>
```

(continued on next page)

```
<!— Check if any results where returned —>
<cfif not isdefined(“OUT_GeoCodeResults”)>
<br>
   <div align=”center”>No features found.</div>
<cfelse>
Records: <b>#OUT_GeoCodeResults.RecordCount#</b><br>
</cfif>
</cfoutput>
</font>
<!— If results were returned, build a table to display results —>
<cfif isdefined(“OUT_GeoCodeResults”)>
<table bgcolor=”000000" cellpadding=3 cellspacing=1 border=1>
<tr>
   <!— Loop through the results to grab column names —>
   <cfloop index=”curCol” list=”#OUT_GeoCodeResults.ColumnList#”>
      <cfoutput>
      <td>
        <font face=”Arial” size=”2" color=”ffffff”><b>#curCol#</b></font>
      </td>
      </cfoutput>
   </cfloop>
</tr>
<!— Loop through results to grab values —>
<cfloop Query=”OUT_GeoCodeResults”>
<tr bgcolor=”ffffff”>
   <cfloop index=”curCol” list=”#OUT_GeoCodeResults.ColumnList#”>
      <cfoutput>
      <td>
        <font face=”Arial” size=”2">&lt;b>#evaluate(curCol)#</b></font>
      </td>
      </cfoutput>
   </cfloop>
</tr>
</cfloop>
</table>
</cfif>
```

Using the ColdFusion Sample Renderer

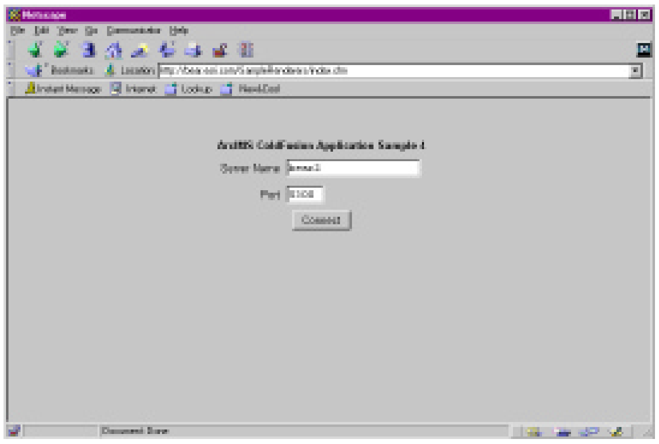
The ColdFusion Sample Renderer is an application that demonstrates techniques that can be used to modify the rendering, or symbology, defined in a MapService. It is not intended to be a complete application but rather a demonstration of the ArcIMS ColdFusion tags as they relate to rendering.

Starting the MapServices

This sample works with any image MapServices running on the host you choose. However, the MapServices must meet this requirement—the Map configuration file (.axl) can't contain layer names with spaces. For example the SanFrancisco MapService used in the template does not work in the renderer. It is based on sf.axl which has a layer named 'Art Galleries'.

Opening the template

In a browser, type http://<local host>/website/ColdFusion/SampleRenderers/index.cfm to open the sample. A form with the Server Name and Port input fields appears.


ArcIMS ColdFusion Application Sample 4

Server Name [illegible]

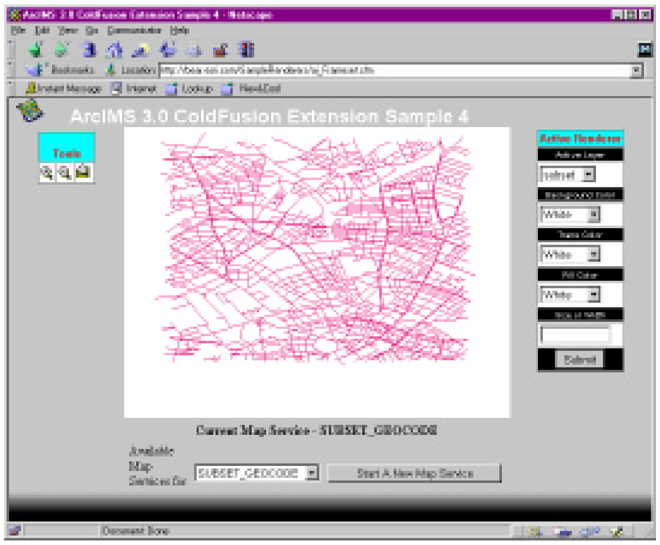
Port [illegible]

Connect

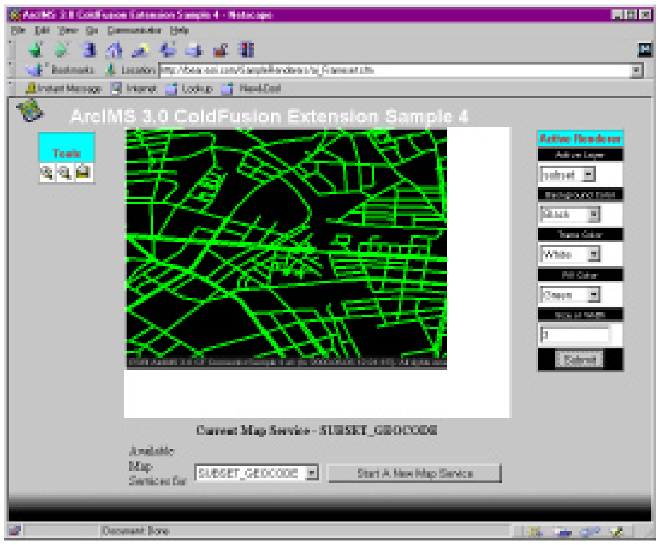
Type in the Server name and port number where the ArcIMS Application Server is running then click Connect.

The main page appears with an area for the map and a scrolling list of MapServices.

Note: If you don't get a list of MapServices on the scrolling list and receive an error about CFX_ESRIMAP, you haven't correctly configured ColdFusion for ArcIMS. Follow the configuration steps outlined in the *ArcIMS 3.0 Installation Guide,* Appendix C, 'Configuring ColdFusion for the ColdFusion Connector', and try connecting to the sample again.



Select the MapService and click Start A New MapService. A list of layers should appear on the right, a toolbar on the left, and a map image in the middle.



Select the active layer, then zoom to the area of interest. Note: The sample tools are inactive after a renderer is submitted, so use the tools to zoom where you want prior to submiting values for new symbology.

Select a new Background Color for the image, a Fill Color for the features in the active layer, and Size or Width for features in the active layer, then click Submit. If your active layer has line or polygon features, the value sets the line width; if points, it sets the point marker size, however point rendering is not available in the sample. The map is rendered with new colors and the copyright information in the acetate layer.

ColdFusion Sample Renderer files

The following files are included in the ColdFusion Sample Renderer. All files are located in the <ArcIMS installation directory>\Samples\ColdFusion\SampleRenderers. See ‘Installing the samples’ earlier in the chapter if you can’t locate the directory. They are briefly introduced here and are discussed in more detail later in this chapter.

- ai_Bottom.cfm—displays the gray graphic at the bottom of the template.
- ai_Frameset.cfm—creates the frame structure of the page and gathers server access information.
- ai_Globals.cfm—contains the CFApplication declaration; sets attributes for server access.
- ai_Map.cfm—generates a new map.
- ai_Note.cfm—positioned below toolbar. Not used in application.
- ai_Renderer.cfm—sets initial rendering properties and constructs new ArcXML to modify rendering of a MapService.
- ai_RendererForm.cfm—displays the user inteface form for rendering properties and sends the form to ai_Renderer.cfm.
- ai_Service.cfm—allows users to select a MapService. This is the first screen of the template.
- ai_Toolbar.cfm—defines structure and contains icons for toolbar. Uses ui.js for toggling functionality.
- ai_Top.cfm—displays the gray header graphic and ArcIMS logo at the top of the template.
- index.cfm—starts the sample renderer application.
- ui.js—JavaScript file containing a function to handle the dynamic toolbar.

ColdFusion Sample Renderer frames

The ColdFusion Sample Renderer is composed of a series of files that open in sequence.The starting page, index.cfm, initializes the application then calls ai_Frameset.cfm which divides the main page into seven frames—TopFrame, ToolFrame, NoteFrame, MapFrame, ServicesFrame, RendererFrame, and BottomFrame.

TopFrame and BottomFrame

The TopFrame and BottomFrame hold the graphics for the top and bottom of the template. Ai_Top.cfm and ai_Bottom.cfm fill theses frames.

ToolFrame

The ToolFrame holds the toolbar for the template. The file ai_Toolbar.cfm fills the ToolFrame.

For many of the tools, when one is clicked it is selected but not executed. Instead, these tools are executed when the Map is clicked or a Submit button is clicked. The JavaScript function switchTool, in ui.js, gets called when a tool is clicked.

NoteFrame

The NoteFrame fills the area below the toolbar.

MapFrame

The MapFrame holds the map for the template. The purpose of the MapFrame is to hold the map image and the hidden inputs passed from the ToolFrame. The file ai_Map.cfm opens the MapFrame.

ServicesFrame

The ServicesFrame is positioned below the map frame and provides the user with a list of MapServices. ai_Services.cfm fills the ServicesFrame.

RendererFrame

The RendererFrame holds the user interface for entering rendering values. The file ai_RendererForm.cfm fills the RendererFrame.

Functionality of the Sample Renderer

In this section you are introduced to the ColdFusion techniques used to implement the functionality of the ColdFusion Sample Renderer presented earlier in the chapter.

Basic framework of this application is the same as in ColdFusion Template. The main difference between the samples is how the map image is generated. The ColdFusion Template uses CF_ARCIMS GENERATEMAP action type, and the ColdFusion Sample Renderer uses CFX_ESRIMAP REQUEST action type.

In this section you examine ai_Renderer.cfm. This file contains the code to perform the rendering functionality. All other files are similar to those in the ColdFusion Template. See 'Functionality of the ColdFusion Template' for information on coding the toolbar, layer manipulation, MapServices selection, and server access.

Ai_Renderer.cfm page uses CFX_ESRIMAP REQUEST action type to modify default renderers for a selected layer, and to put a text description on the acetate layer. Although CFX_ESRIMAP is used in this example, CF_ARCIMS request could be used without any other change in the code.

Initializing the renderer values

The file starts by checking if the renderer variables are defined and specifying defaults if they are not. Renderer variables for foreground color, background color, and symbol size are in the form variables defined in the ai_RenderForm.cfm page if the user initiated this page by clicking Submit.

```
<!--
   ai_Renderer.cfm - generates customized map
->
<cfinclude template="ai_Globals.cfm">
<cfif isDefined("Form.BackColor")>
   <cfset curBackColor = Form.BackColor >
<cfelse>
   <cfset curBackColor = "255,255,0" >
</cfif>
<cfif isDefined("Form.TransColor")>
   <cfset curTransColor = Form.TransColor >
<cfelse>
   <cfset curTransColor = "255,255,0" >
</cfif>
<cfif isDefined("Form.FillColor")>
   <cfset curFillColor = Form.FillColor >
<cfelse>
   <cfset curFillColor = "0,255,255" >
</cfif>
```

(continued on next page)

```
<cfif isDefined("Form.Size")>
   <cfset curSize = Form.Size >
<cfelse>
   <cfset curSize = "12" >
</cfif>
<CFSET myRed="255,0,0">
<CFSET LINE="line">
<CFSET POINT="point">
<CFSET POLYGON="polygon">
<CFSET myLocation="5,5">
<CFSET copyrightText="ESRI ArcIMS 3.0 CF Connector Sample 4 at: #now()#">
<CFSET myLabel="You are customizing on layer: #Layername#">
```

Creating the ArcXML code for rendering

After the variables are defined, you assemble the ArcXML code for the request. You create the ArcXML request by appending ArcXML code to an AXLInput ColdFusion variable. In this phase, it is important to distinguish ColdFusion tags such as CFSET that are part of page code, and ArcXML tags that are combined into the string that is sent to the server. Because the ArcXML tags are quoted, they are invisible to ColdFusion. As with all requests that create a map image, you use the ArcXML REQUEST and GET_IMAGE tags. If you are unfamiliar with ArcXML, see the *ArcXML Programmer's Reference Guide*.

First, you create the XML header and open the ARCXML, REQUEST, GET_IMAGE and PROPERTIES tags. Background color and image size are specified using subtags of PROPERTIES tag. At the end, open the LAYERLIST tag.

```
<CFSET AXLInput = '<?xml version="1.0"?><ARCXMLversion="1.0.1">
<REQUEST><GET_IMAGE><PROPERTIES>'>

<CFSET AXLInput = #AXLInput# & '<BACKGROUND color="' & #curBackColor# & '"
transcolor="' & #curTransColor# & '" /> '>

<CFSET AXLInput = #AXLInput# & '<IMAGESIZE height="' &
#Session.LayerInfo.ImageHeight# & '" width="' & #Session.LayerInfo.ImageWidth# &
'" /> '>

<CFSET AXLInput = #AXLInput# & '<ENVELOPE minx="' & #Session.LayerInfo.minx# & '"
miny="' & #Session.LayerInfo.miny# & '" maxx="' & #Session.LayerInfo.maxx# & '"
maxy="' & #Session.LayerInfo.maxy# & '" /> <LAYERLIST>'>
```

Now you are ready to loop through the list of layers and modify the renderer for the active layer. The active layer is selected from ai_RendererForm.cfm and its ID is set in the selectLayerID variable. To modify the active layer renderer, use the LAYERDEF tag. It is required that the LAYERDEF tag always be inside the LAYERLIST tag. The loop shown below opens the LAYERDEF tag for the active layer id.

```
<!– set active layer visible=true –>
<cfloop Query=”Session.LayerInfo.LayerTable”>
   <CFLOOP INDEX=”index” LIST=”#Session.LayerInfo.LayerTable.ID#”>
       <CFIF #index# eq #selectLayerID# >
        <CFSET AXLInput = #AXLInput# & ‘<LAYERDEF id=”’ & #index# & ‘“ type=”’&
        #selectLayerType# & ‘“ visible=”true” >’>
      </cfif>
   </CFLOOP>
</cfloop>
```

With the LAYERDEF tag open, you can append renderer tags. Symbol tags, such as SIMPLEMARKERSYMBOL, SIMPLELINESYMBOL or SYMPLEPOLYGONSYMBOL, are chosen based on layer type. Symbol attributes of color and size are set from variables that are originally defined in the form in ai_Renderer.cfm page.

```
<CFSET AXLInput = #AXLInput# & ‘<SIMPLERENDERER>’>
<cfif #selectLayerType# eq #POINT# >
   <CFSET AXLInput = #AXLInput# & ‘<SIMPLEMARKERSYMBOL color=”’ & #curFillColor# &
   ‘“ type=”Circle’ & ‘“ size=”’ & #curSize# & ‘“ /> ‘>
<cfelseif #selectLayerType# eq #LINE#>
   <CFSET AXLInput = #AXLInput# & ‘<SIMPLELINESYMBOL  color=”’ & #curFillColor# &
   ‘“ type=”SOLID’ & ‘“ width=”’ & #curSize# & ‘“ /> ‘>
<cfelseif #selectLayerType# eq #POLYGON#>
   <CFSET AXLInput = #AXLInput# & ‘<SIMPLEPOLYGONSYMBOL  fillcolor=”’ &
   #curFillColor# & ‘“ filltype=”lightgray’ & ‘“ boundarywidth=”’ & #curSize# &
   ‘“ boundarycolor=”’ & #myRed# & ‘“ /> ‘>
</cfif>
```

To finalize the ArcXML code that modifies the layer renderer, close the SIMPLERENDERER, LAYERDEF, LAYERLIST, and PROPERTIES tags.

```
<CFSET AXLInput = #AXLInput# & ‘</SIMPLERENDERER></LAYERDEF>’>
<CFSET AXLInput = #AXLInput# & ‘</LAYERLIST></PROPERTIES>’>
```

Creating the ArcXML code for the acetate layer

Acetate layer objects are defined in a new LAYER tag. The LAYER tag adds new content to the MapService and is a subtag of the GET_IMAGE tag. It is important to distinquish the LAYER tag from the LAYERDEF. The LAYERDEF tag modifies content that is already in the MapService and it is a subtag of the PROPERTIES tag.

There are two objects on the acetate layer—a polygon that is used as text background and the text.

```
<!– Add Acetate Layer –>
<CFSET AXLInput = #AXLInput# &
                    ‘<LAYER type=”Acetate” name=”#Layername#” visible=”true”>’ &
                    ‘<OBJECT units=”pixel”>’ &
                    ‘<POLYGON coords=”0,0,#Session.LayerInfo.ImageWidth#,0,
                    #Session.LayerInfo.ImageWidth#,15,0,15,0,0">
                    <SIMPLEFILLSYMBOL color=”35,35,35" boundary=”true”/>
                    </POLYGON>’ &
                    ‘</OBJECT><OBJECT units=”pixel”>’ &
                    ‘<TEXT coord=”4,4" label=”’ & #copyrightText# & ‘. All rights
                    reserved.”>’ &
                    ‘<TEXTMARKERSYMBOL halignment=”right” valignment=”top”
                    font=”Arial” fontcolor=”204,204,204" antialiasing=”true”
                    size=”9" />’ & ‘</TEXT></OBJECT></LAYER>’>
```

To complete the ArcXML request, close the GET_IMAGE, REQUEST and ARCXML tags.

```
<CFSET AXLInput = #AXLInput# & ‘</GET_IMAGE></REQUEST></ARCXML>’>
```

Sending the ArcXML request

The request is ready to be sent. Use the CFX_ESRIMAP tag with the action REQUEST. You could also use the CF_ARCIMS tag with identical syntax. Refer to Chapter 2, topic ‘Comparison of CF_ARCIMS and CFX_ESRIMAP’.

```
  <!– send AXL request to the server –>
    <CFX_ESRIMAP ACTION=”REQUEST”
    SERVICENAME=”#Session.ServiceName#”
    SERVERNAME=”#Session.ServerName#”
    SERVERPORT=”#Session.ServerPort#”
    CUSTOMSERVICE=””
    AXLTEXT=”#AXLInput#”
    GENERATEHTML=”false”>
```

Checking the ArcXML response

This code checks if any response is received.

```
<CFIF NOT #ParameterExists(OUT_RESPONSEAXL)# >
    <!– ArcIMS not running –>
    <CFOUTPUT>No answer from ArcIMS<br></CFOUTPUT>
<CFELSEIF #OUT_RESPONSEAXL# IS “”>
    <!– ArcIMS has a problem –>
    <CFOUTPUT>ArcIMS answer is empty<br></CFOUTPUT>
```

If there is a response, it checks for an error, and if found, prints the response.

```
<CFELSEIF Find("<ERROR", #OUT_RESPONSEAXL#, 1) IS NOT 0>
      <CFOUTPUT>#HTMLEditFormat(OUT_RESPONSEAXL)#<br></CFOUTPUT>
```

The ColdFusion function HTMLEditFormat() is used to print out the response. The HTMLEditFormat() function is used because it escapes all special characters and presents ArcXML code as normal text. It is not advisable to directly include the ArcXML code response from the ArcIMS server to a CFM or HTML page because it may confuse either the ColdFusion server or client browser.

If you want more advanced error trapping, you can further parse the response. The code below that is commented out parses the response and extracts the exact error message from the response string.

```
<!--
      <CFSET posBeg = Find("<ERROR", #OUT_RESPONSEAXL#, 1)>
      <CFSET posStart = Find(">", #OUT_RESPONSEAXL#, #posBeg#) + 1>
      <CFSET posEnd = Find("</ERROR>", #OUT_RESPONSEAXL#, 1)>
      <CFSET msg = Mid(OUT_RESPONSEAXL, posStart, posEnd-posStart)>

      <CFSET posStart = Find(" CODE", #OUT_RESPONSEAXL#, 1) + 7>
      <CFIF posStart IS not 7>
        <CFSET posEnd = Find('"', #OUT_RESPONSEAXL#, #posStart#)>
        <CFSET code = Mid(OUT_RESPONSEAXL, posStart, posEnd-posStart)>
      <CFELSE>
        <CFSET code = "">
      </CFIF>
      <CFOUTPUT>ArcIMS Error: #msg# (#code#)<br></CFOUTPUT>
->
```

As an alternative to writing this code, you can ask the connector to parse the response. If you include PARSERESPONSEAXL='true' attribute in the CFX_ESRIMAP tag, the connector creates the OUT_ERROR variable and sets its value to the error message.

Some server errors cause CFX_ESRIMAP to throw an exception and no output variables are defined. To catch such situations you might try ColdFusion exception handling procedures CFTRY and CFCATCH. In this sample they are not used.

If there is no error in the response, it is parsed, looking for the resulting URL and extent attributes. If you use PARSERESPONSEAXL='true' attribute in the CFX_ESRIMAP tag, the connector creates the variables for the image url and extents—OUT_IMAGEURL, OUT_MINX, OUT_MINY, OUT_MAXX, OUT_MAXY. See Chapter 2, topic 'GENERATEMAP'.

```
<CFELSE>
     <CFSET posStart = Find(“ url”, #OUT_RESPONSEAXL#, 1) + 6>
     <CFSET posEnd = Find(‘“‘, #OUT_RESPONSEAXL#, posStart)>
     <CFSET mapURL = Mid(#OUT_RESPONSEAXL#, #posStart#, #posEnd#-#posStart#)>
     <CFOUTPUT><IMG src=#mapURL#><br></cfoutput>

     <!— output map extent —>
     <CFSET posBegin = Find(“ENVELOPE”, #OUT_RESPONSEAXL#, 1)>

     <CFSET posStart = Find(“minx”, #OUT_RESPONSEAXL#, #posBegin#) + 6>
     <CFSET posEnd = Find(‘“‘, #OUT_RESPONSEAXL#, posStart)>
     <CFSET OUT_MinX = Mid(#OUT_RESPONSEAXL#, #posStart#, #posEnd#-#posStart#)>

     <CFSET posStart = Find(“miny”, #OUT_RESPONSEAXL#, #posBegin#) + 6>
     <CFSET posEnd = Find(‘“‘, #OUT_RESPONSEAXL#, posStart)>
     <CFSET OUT_MinY = Mid(#OUT_RESPONSEAXL#, #posStart#, #posEnd#-#posStart#)>

     <CFSET posStart = Find(“maxx”, #OUT_RESPONSEAXL#, #posBegin#) + 6>
     <CFSET posEnd = Find(‘“‘, #OUT_RESPONSEAXL#, posStart)>
     <CFSET OUT_MaxX = Mid(#OUT_RESPONSEAXL#, #posStart#, #posEnd#-#posStart#)>

     <CFSET posStart = Find(“maxy”, #OUT_RESPONSEAXL#, #posBegin#) + 6>
     <CFSET posEnd = Find(‘“‘, #OUT_RESPONSEAXL#, posStart)>
     <CFSET OUT_MaxY = Mid(#OUT_RESPONSEAXL#, #posStart#, #posEnd#-#posStart#)>
</CFIF>
```

Debugging tip for ArcXML request and response

The ai_Renderer.cfm file ends with a piece of code that can be used for debugging. It displays the actual request sent to the server, and the response received from it.

```
<!– IF YOU WANT TO SEE WHAT EXACTLY AXL REQUEST SEND TO
    OR RESPONSE FROM THE SERVER, USE THE FOLLOWING CODE.
–>
<!– <cfoutput>
   <cfif isDefined(“IN_RequestAXL”)>
   <b>Request AXL:</b><br>
   <pre>
   #HtmlEditFormat(IN_RequestAXL)#
   </pre>
   </cfif>
   <cfif isDefined(“OUT_ResponseAXL”)>
   <b>Response AXL:</b><br>
   <pre>
   #HtmlEditFormat(OUT_ResponseAXL)#<br>
   </pre>
   </cfif>
</cfoutput>  –>
```

ArcIMS ColdFusion Tag Reference

2

IN THIS CHAPTER

- **How ArcIMS and ColdFusion work together**
- **ArcIMS in ColdFusion Studio**
- **ArcXML ColdFusion tag reference**
- **Application development and debugging tips**

Note

For information on how to install and set up the ArcIMS ColdFusion Connector, see the ArcIMS 3.0 Installation Guide.

This chapter provides a reference to the ArcIMS ColdFusion tags available with the ColdFusion Connector, as well as some application development and debugging tips. It also show how ArcIMS is incorporated into the ColdFusion Studio interface.

This chapter assumes a familiarity with HTML, ColdFusion tags, and ArcXML. If you are unfamiliar with ArcXML, see the *ArcXML Programmer's Reference Guide*.

How ArcIMS and ColdFusion work together

The ArcIMS ColdFusion Connector brings map publishing and GIS capabilities to the ColdFusion Server. Using ColdFusion Studio, web developers can create geographic web pages in a fraction of time that would be required using other technologies such as JavaScript or client Java programming. Moreover, since all of the processing is done at the server, the web site is less dependent on client browsers and less demanding on client system and network, making it a very good solution for Internet map publishing.

In the ArcIMS Architecture ColdFusion connector replaces default servlet connector. Instead of communicating with the servlet using low-level ArcXML requests, the developer can concentrate on the program logic, and use high level Cold Fusion Markup Language (CFML) tags. ArcIMS mapping functions are accessed from CFML using custom ArcIMS connector tags.

ArcIMS custom ColdFusion tags and attributes

All of basic mapping and GIS functions are performed using the <CF_ARCIMS> custom tag and its subtags:

<CF_ARCIMS_LAYER>—specifies layer visibility.

<CF_ARCIMS_SQL>—contains the WHERE clause of the SQL statement.

<CF_ARCIMS_MULTIPOINT>—defines points drawn on the acetate layer.

<CF_ARCIMS_POLYLINE>—defines polylines drawn on the acetate layer.

<CF_ARCIMS_POLYGON>—defines polygons drawn on the acetate layer.

The details of the tags are specified in the tag attributes. All of the tags have these attributes.

SERVERNAME—specifies the name of the host that runs ArcIMS Application Server.

SERVERPORT—specifies the port number that the ArcIMS server is listening on.

GENERATEHTML—indicates whether the tag should tag generate HTML code with the result.

ACTION—sets the type of the action to be performed.

<CF_ARCIMS> ACTION attributes

There are several possible values for the ACTION attribute that perform different operations. A short description is found below and a complete reference to each value is described later in the chapter.

GENERATEMAP—generates a map image

LEGEND—generates a legend image

QUERY—performs attribute, spatial, or combined query to the database

IDENTIFY—identifies features at specified coordinates

REQUEST—executes any kind of a request specified by ArcXML code

GEOCODE—geocodes an address

EXTRACT—extract features from ArcIMS layers, and creates a shapefile

GETMAPSERVICES—returns a list of MapServices running on the server

GETSERVICEINFO—returns information about specific MapService

There are two main ways to use CF_ARCIMS tag—with a true or false setting of the GENERATEHTML attribute.

When the GENERATEHTML attribute is set to true the CF_ARCIMS tag generates the appropriate HTML code to present the result. For example, if the action is GENERATEMAP, the HTML <IMG> tag generated on the result page points to the new image, and the image appears on the page in the browser; if the action is QUERY, an HTML table is created. This frees the developer from writing HTML code to present the result.

When GENERATEHTML is set to false, no HTML code is generated. All required information is placed in CFML variables, and a developer can use them to present the data. For example, if the request is GENERATEMAP, the CF_ARCIMS tag creates a variable OUT_IMAGEURL that contains the URL of the map image generated by the ArcIMS Spatial Server. That variable can then be used inside HTML <IMG> or <INPUT> tag to display the image.

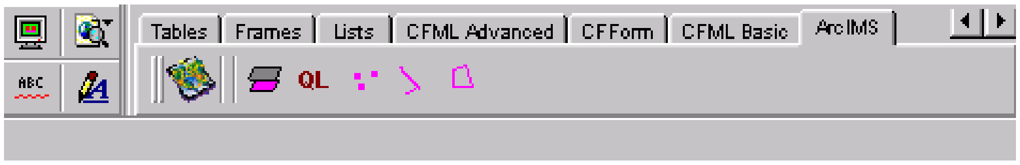
Some requests generate CFML queries that you can use in <cfoutput>, <cfloop>, or any other CFML tag or function that works with CFML queries. For example, the QUERY action creates OUT_QUERYTABLE that contains all of the records returned by the query. You can use <cftable> or <cfoutput> to present the result. An example of this is found in the QUERY topic.

It is important to note that, with the exception of using the REQUEST action, all ArcXML processing is completely hidden from the ColdFusion developer. The developer only works with high-level CFML tags, while the ColdFusion Connector does the ArcXML creation and parsing of the response returned by the ArcIMS server. For more information on the REQUEST action see its reference pages and 'Application development and debugging tips' later in the chapter.

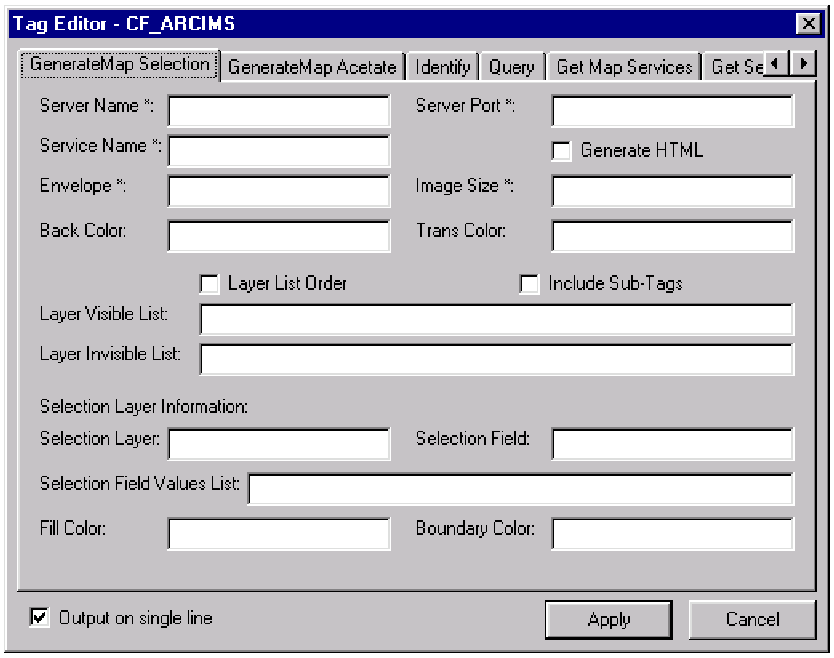
ArcIMS in ColdFusion Studio

Perhaps the simplest way to learn the tags and attributes that are defined by the ArcIMS ColdFusion Connector is through the ColdFusion Studio interface. After installing the connector, ColdFusion Studio contains a new toolbar for ArcIMS.

The toolbar contains six tools for working with the new ArcIMS tags: <CF_ARCIMS>, <CF_ARCIMS_LAYER>, , <CF_ARCIMS_SQL>, <CF_ARCIMS_MULTIPOINT>, <CF_ARCIMS_POLYLINE>, and <CF_ARCIMS_POLYGON>.



When you choose the <CF_ARCIMS> tool, a dialog open presenting all the tag attributes.



The tabs across the top represent the different values of the ACTION attribute of the <CF_ARCIMS> tag. For example, the tag code for the GenerateMap Selection might look like this.

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.46,37.76,-122.42,37.79"
   IMAGESIZE=”400,400"
   LAYERLISTORDER=”true”
   GENERATEHTML=”true”>
</CF_ARCIMS>
```

You can use ColdFusion Studio to give you an introduction to the new custom tags for ArcIMS and to help build your applications. See the following section for a detailed description of the ACTION attribute values and subtags.

GENERATEMAP

Purpose

Generates a map image based on an Image MapService.

Syntax

Acetate layer objects and layer visibility can be specified using either CF_ARCIMS tag attributes or subtags.

Basic syntax

```
<CF_ARCIMS ACTION = "GENERATEMAP"
   SERVICENAME = "service name"
   SERVERNAME="server name"
   SERVERPORT="server port"
   ENVELOPE="minx,miny,maxx,maxy"
   IMAGESIZE="width,height"
   LAYERLISTORDER="TRUE|FALSE"
   LAYERVISIBLELIST= "layer_id1, layer_id2, layer_id3…"
   LAYERINVISIBLELIST= "layer_id1, layer_id2, layer_id3…"
   BACKGROUNDCOLOR="R,G,B"
   TRANSPARENCYCOLOR="R,G,B"
   SELECTIONLAYER="selected layer name"
   SELECTIONFIELD="selected field name"
   SELECTIONFIELDVALUESLIST="ID1, ID2, ID3, …"
   SELECTIONFILLCOLOR="R,G,B"
   SELECTIONBOUNDARYCOLOR="R,G,B"
   ADDPOINTS="X1,Y1,X2,Y2"
   POINTCOLOR="R,G,B"
   ADDLINES="x1,y1,x2,y2,x3,y3 ; x4,y4,x5,y5,x6,y6"
   LINECOLOR="R,G,B"
   ADDPOLYGONS="x1,y1,x2,y2,x3,y3,x4,y4 ; x5,y5,x6,y6,x7,y7"
   POLYGONCOLOR="R,G,B, R,G,B"
   ADDIMAGES="image icon path"
   IMAGESPOSITION="x1,y1,x2, y2, x3, y3"
   LINEWIDTH="value"
   POINTWIDTH="value"
   POLYGONWIDTH="value"
   ADDTEXT="STRING1, STRING2, STRING3"
   TEXTSTARTPOSITION="X1,Y1, X2, Y2, X3, Y3"
   TEXTFONT="FONTNAME,SIZE,STYLE,FONTCOLOR(R),FONTCOLOR(G),FONTCOLOR(B)"
   TEXTBACKGROUNDCOLOR="R,G,B"
   TEXTROTATIONANGLE="Angle of rotation in degrees"
   TEXTSHADOW ="R,G,B"
   TEXTGLOW="R,G,B"
   GENERATEHTML="TRUE|FALSE">
```

Extended syntax

Specify layer visibility using the CF_ARCIMS_LAYER subtag as shown below instead of LAYERVISIBLELIST or LAYERINVISIBLELIST attributes shown in the basic syntax example. Use one or more CF_ARCIMS_LAYER subtags to define layers inside the CF_ARCIMS tag.

```
   <CF_ARCIMS_LAYER  LayerID=”Layer_ID”  Visible=[true|false] >
```

Specify acetate layer objects using the following subtags instead of acetate layer attributes shown on the basic syntax. For example, use CF_ARCIMS_MULTIPOINT instead of ADDPOINTS, CF_ARCIMS_POLYLINE instead of ADDLINES, CF_ARCIMS_POLYGONS instead of ADDPOLYGONS, and CF_ARCIMS_TEXT instead of ADDTEXT. Many acetate layer object tags can be combined inside one CF_ARCIMS tag.

```
<CF_ARCIMS_MULTIPOINT>
   <Point x=”x1” y=”y1”/>
   <Point x=”x2” y=”y2”/>
   <Point x=”x3” y=”y3”/>
</CF_ARCIMS_MULTIPOINT>

<CF_ARCIMS_POLYLINE>
   <path><Point x=”x1” y=”y1”/><Point x=”x2” y=”y2”/><point x=”x3” y=”y3”/></path>
   <path><Point x=”x4” y=”y4”/> <Point x=”x5” y=”y5”/></path>
</CF_ARCIMS_POLYLINE>

<CF_ARCIMS_POLYGON>
   <RING>
   < Point x=”x1” y=”y1”/><Point x=”x2” y=”y2”/><Point x=”x3” y=”y3”/>
   </RING>
   <RING>
   <Point x=”x4” y=”y4”/><Point x=”x5” y=”y5”/><Point x=”x6” y=”y6”/>
   </RING>
</CF_ARCIMS_POLYGON>

<CF_ARCIMS_TEXT>
   <TEXT String=”value” Position=”x,y”/>
   <TEXT String=”value” Position=”x,y”/>
</CF_ARCIMS_TEXT>
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
ADDIMAGES	N	String	N/A	N/A	Path or URL of image to draw on the map.
ADDLINES	N	Doubles	N/A	N/A	List of coordinates in image coordinate system (pixels) of lines to be drawn on an acetate layer. Lines are separated by a semicolon, so list format is: x11,y11,x12, y12;x21,y21,x22,y22, ...
ADDPOINTS	N	Doubles	N/A	N/A	List of point coordinates to draw on the acetate layer (x1,y1,x2,y2, ...) in image coordinates (pixels).
ADDPOLYGONS	N	Doubles	N/A	N/A	List of coordinates (x1, y1 x2, y2, ...) in image coordinate system (pixels) of polygons to draw on an acetate layer. Polygons are defined by rings separated by a semicolon, so list format is: x11,y11, x12, y12;x21,y21,x22,y22, ...
ADDTEXT	N	String	N/A	N/A	Acetate layer text to draw on the map. Separate with commas to render multiple strings.Text is placed at location specified by TEXTSTARTPOSITION.
BACKGROUNDCOLOR	N	Color	From MapService	0,0,0– 255,255,255	Map image background color.
ENVELOPE	Y	Doubles	N/A	N/A	Extent of map to be generated in map units as a comma-separated list (MinX, MinY, MaxX, MaxY).
GENERATEHTML	N	Boolean	False	True, False	If true, HTML IMG tag is generated in place of CF_ARCIMS during run time. If false, all variables are set but no tag is generated.
IMAGESPOSITION	N	Doubles	N/A	N/A	Coordinates (in pixels) where images should be placed (x1, y1, x2, y2,...). First pair is used for first image in ADDIMAGE list, second for second image, and so on.
IMAGESIZE	Y	Doubles	N/A	N/A	Size of map image to be generated (width, height) in pixels.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
LAYERINVISIBLE LIST	N	Integer	N/A	N/A	List of IDs for layers to turn on.
LAYERLISTORDER	N	Boolean	False	True, False	If true, layers are drawn in the order listed in LAYERVISIBLE LIST, and only layers specified in LAYERVISIBLELIST are used. If false, layers are drawn in the order they appear in the map file, and all visible layers are used.
LAYERVISIBLE LIST	N	Integer	N/A	N/A	List of IDs for layers to turn off.
LINECOLOR	N	Color	N/A	0,0,0–255,255,255	Color to render acetate layer lines.
LINEWIDTH	N	Integer	1	N/A	Width of acetate layer lines.
POINTCOLOR	N	Color	N/A	0,0,0–255,255,255	Color to render acetate layer points.
POINTWIDTH	N	Integer	8	N/A	Size of acetate layer points in pixels.
POLYGONCOLOR	N	Colors	N/A	0,0,0–255,255,255	Two colors to render acetate layer polygons. First RGB value is polygon fill color and second RGB value is polygon outline color.
POLYGONWIDTH	N	Integer	1	N/A	Width of acetate layer polygon outline.
SELECTIONBOUN DARYCOLOR	N	Color	255,0,0	0,0,0–255,255,255	Color to outline highlighted features.
SELECTIONFIELD	N	String	N/A	N/A	Name of field to select features to highlight.
SELECTIONFIELD VALUESLIST	N	String	N/A	N/A	List of values to select features to highlight.
SELECTIONFILLCO	N	Color	255,0,0	0,0,0–255,255,255	Color to render highlighted features.
SELECTIONLAYER	N	String	N/A	N/A	ID of the layer containing features to highlight.
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to be drawn.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
TEXTBACKGROUND COLOR	N	Color	N/A	0,0,0–255,255,255	Text background color of acetate layer text.
TEXTFONT	N	String	N/A	N/A	Font to draw acetate layer text on the map. String is comprised of font name, font size, font style, font color. Font name must exist on Spatial Server. Font size is in points. Font styles are Regular, Bold, Italic, Underline, or Outline. Font color is RGB specification of the text color.
TEXTGLOW	N	Color	N/A	0,0,0–255,255,255	Acetate layer text glow color. Text can have shadow, glow, or none, but not both.
TEXTROTATION ANGLE	N	Integer	N/A	0–360	Angle in degrees used to rotate the acetate layer text.
TEXTSHADOW	N	Color	N/A	0,0,0–255,255,255	Acetate layer text shadow color.
TEXTSTARTPOSITION	N	Doubles	N/A	N/A	Coordinates (in pixels) where acetate layer text should be placed on map (x1,y1,x2,y2,...). First coordinate pair is used for first string in ADDTEXT, second for the second, and so on.
TRANSPARENTCOLOR	N	Color	From MapService	0,0,0–255,255,255	Color to render imageas transparent.

Subtags

Name	Required	Occurrences	Notes
CF_ARCIMS_LAYER	N	Many	Used to specify layer visibility. There should be one CF_ARCIMS_LAYER for every layer that needs to be turned on or off. If LAYERLISTORDER is false, only layers with visibility different from the default needs to be specified. If LAYERLISTORDER is true, only layers specified with CF_ARCIMS_LAYER are rendered. All layers that need to be visible should be specified regardless of the default visibility. Replaces LAYERVISIBLELIST and LAYERINVISIBLELIST attributes.

Name	Required	Occurrences	Notes
CF_ARCIMS_MULTIPOINT	N	One	Defines points to be drawn on the acetate layer. Multiple points can be specified using the POINT tag inside one CF_ARCIMS_MULTIPOINT tag. If used, specify point size and color with POINTCOLOR and POINTWIDTH attributes. Replaces ADDPOINTS attribute.
CF_ARCIMS_POLYGON	N	One	Defines polygons to be drawn on the acetate layer. Polygons are specified using RING subtags. Only polygons without holes are supported for an acetate layer; the subtag HOLE is not permitted. Replaces ADDPOLYGONS attribute.
CF_ARCIMS_POLYLINE	N	One	Defines polylines to be drawn on the acetate layer. Multiple lines can be specified using the PATH tag inside CF_ARCIMS_POLYLINE. Each PATH tag can have two or more POINT tags. Replaces ADDLINES attribute.
CF_ARCIMS_TEXT	N	One	Defines text to be drawn on the acetate layer. Text is specified using the TEXT subtag. The TEXT tag contains String and Position attributes. There can be multiple TEXT tags inside CF_ARCIMS_TEXT. Replaces ADDTEXT attribute.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_IMAGEPATH	Y	String	Path of images generated by the Spatial Server as seen by the Spatial Server.
OUT_IMAGEURL	Y	String	URL of images generated by the Spatial Server.
OUT_MAXX	Y	Integer	Envelope of the returned map in map coordinates. Coordinates can be different from those in the ENVELOPE attribute in request because of the difference in the aspect ratio between map extent and image size. OUT_MAXX is X-coordinate of the upper-right corner.
OUT_MAXY	Y	Integer	See usage for OUT_MAXX. OUT_MAXY is Y-coordinate of the upper-right corner.

Variable Syntax	Always Generated	Variable Type	Usage
OUT_MINX	Y	Integer	See usage for OUT_MAXX. OUT_MINX is X-coordinate of the lower-right corner.
OUT_MINY	Y	Integer	See usage for OUT_MAXX. OUT_MINY is Y-coordinate of the lower-right corner.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

- Since GENERATEMAP sends the request to the Spatial Server to generate a map image, it can be used only with Image MapServices. Feature MapServices can't be rendered using the ColdFusion Connector.
- This tag exposes just part of the functionality of the ArcIMS Image MapServer. To use more advanced functionality, such as changing symbols, use CF_ARCIMS tag with the REQUEST action.

Notes

- GENERATEMAP is used for all map display functions. Zoom in, zoom out, pan, turn layers on or off, and highlight features functions use GENERATEMAP. For any kind of query (including a spatial query), use the QUERY tag. For spatial queries, the QUERY tag is often combined with a map image generated by GENERATEMAP.
- Layers and symbols used in the map depend on the MapService definition. If a layer is turned off, it does not appear on the map unless it is on the LAYERVISIBLELIST. If a layer is invisible at the current scale, it will not appear even if it is on the LAYERVISIBLELIST. Layers that are turned on and visible at the current scale are rendered unless they are on the LAYERINVISIBLELIST. The only exception is if LAYERLISTORDER is true; in that case, only the layers in LAYERVISIBLELIST are drawn, and no other layer is used regardless of the MapService settings.
- The map extent must be specified, even if the MapService has a default extent. To use the default extent, get the extent using CF_ARCIMS GETSERVICEINFO and use that extent for the GENERATEMAP request. The same is true for the image size.
- The map extent returned from the Spatial Server is often different from the one requested because of the difference in aspect ratio between the map and image. Output variables contain the extent of the generated map.

- The SELECTION attributes provide a way to highlight selected features. However, to highlight them, the Spatial Server must be able to find selected features. The Spatial Server is stateless, and does not keep track the selected set. Since it is not possible to keep the selection on the Spatial Server, the ColdFusion client application must track the selected features in session variables. This can be done only if the layer has a field with unique values for every row or a unique key. The typical procedure to select and highlight features involves using the QUERY tag to retrieve selected features including some unique identifier field, creating a list from the result query key field using a ColdFusion function like valueList() or quotedValueList() and submitting results to GENERATEMAP. The SELECTIONLAYER attribute specifies a layer that contains feature(s) to be highlighted. Only features from one layer can be highlighted per request. SELECTIONFIELD contains the name of the key field, and SELECTIONFIELDVALUELIST contains a list of values. The Spatial Server highlights features that have values from the list in the field. Values in SELECTIONFIELD do not have to be unique. It is not possible to select features based on more than one field or use any other operator except IN.
- All colors are specified as RGB values in a comma-separated list of three integers (0–255) specifying red, green, and blue values, respectively.

Examples

1) To draw a simple map:

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.54,37.65,-122.32,37.84"
   IMAGESIZE=”400,400"
   GENERATEHTML=”true”/>
```

Since GENERATEHTML is set to true, this tag creates the following IMG HTML tag:

```
<IMG SRC=”http://bear/output/SanFrancisco_BEAR902562.png” height=400 width=400 >
```

2) This example demonstrates how to turn a layer on and how to use the resulting OUT_IMAGEURL variable. The layer to be turned on is Zipcodes (layer ID of 1). It uses OUT_IMAGEURL to create the form with the input type image tag.

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.54,37.65,-122.32,37.84"
   IMAGESIZE=”400,400"
   LAYERVISIBLELIST=”1"
   GENERATEHTML=”false”/>
<cfoutput>
<form action=”” method=”POST”>
   <input type=”Image” name=”Map_Image” src=”#OUT_IMAGEURL#” width=”400"
height=”400">
</form>
</cfoutput>
```

Since GENERATEHTML is false, no code is generated by the CF_ARCIMS tag. The output variable #OUT_IMAGEURL# is used in the input tag instead.

The resulting HTML code is:

```
<form action=”” method=”POST”>
   <input type=”Image” name=”Map_Image” src=”http://bear/output/
SanFrancisco_BEAR902563.png” width=”400" height=”400">
</form>
```

3) The following example uses CF_ARCIMS_LAYER subtags with LAYERLISTORDER set to true. The output image only has the layers listed in the tags.

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.46,37.76,-122.42,37.79"
   IMAGESIZE=”400,400"
   LAYERLISTORDER=”true”
   GENERATEHTML=”true”>

   <CF_ARCIMS_LAYER  LayerID=”2"  Visible=”true”>
   <CF_ARCIMS_LAYER  LayerID=”3"  Visible=”true”>
   <CF_ARCIMS_LAYER  LayerID=”5"  Visible=”true”>

   </CF_ARCIMS>
```

This creates a map image with only layers 2, 3, and 5 turned on.

4) This example creates a map image with some acetate layer objects.

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.46,37.76,-122.42,37.79"
   IMAGESIZE=”400,400"
   ADDPOLYGONS=”50,50,100,50,50,100;200,250,300,250,300,350"
   POLYGONCOLOR=”255,200,200,255,0,0"
   POLYGONWIDTH=”6"
   ADDTEXT=”Some text,Some other text”
   TEXTSTARTPOSITION=”100,300,15,250"
   TEXTFONT=”Times,20,Bold,255,255,100"
   TEXTSHADOW=”200,200,200"
   GENERATEHTML=”true” />
```

This creates a map image with two polygons and two text strings.

5) The same request can be written using acetate layer object subtags.

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.46,37.76,-122.42,37.79"
   IMAGESIZE=”400,400"
   POLYGONCOLOR=”255,200,200,255,0,0"
   POLYGONWIDTH=”6"
   TEXTSTARTPOSITION=”100,300,150,250"
   TEXTFONT=”Times,40,Bold,255,255,100"
   TEXTSHADOW=”200,200,200"
   GENERATEHTML=”true” >
```

(continued on next page)

```
   <CF_ARCIMS_POLYGON>
      <Ring>
        <point x=”50" y=”50" />
        <point x=”100" y=”50" />
        <point x=”50" y=”100" />
      </Ring>
      <Ring>
        <point x=”200" y=”250" />
        <point x=”300" y=”250" />
        <point x=”300" y=”350" />
      </Ring>
    </CF_ARCIMS_POLYGON>

    <CF_ARCIMS_TEXT>
      <Text String=”Some text” Position=”100,300" />
      <Text String=”Some other text” Position=”150,250" />
    </CF_ARCIMS_TEXT>

</CF_ARCIMS>
```

LEGEND

Purpose

Creates an image with a map legend for a specific MapService.

Syntax

Layer visibility can be specified using either CF_ARCIMS tag attributes or subtags.

Basic syntax

```
<CF_ARCIMS ACTION = "LEGEND"
   SERVICENAME = "service name"
   SERVERNAME="server name"
   SERVERPORT="server port"
   LAYERLISTORDER="TRUE|FALSE"
   LAYERVISIBLELIST= "id1, id2, , id3…"
   LAYERINVISIBLELIST= "id1, id2, , id3…"
   TITLE="title of the legend"
   FONT="font name, title font size, layer font size, value font size"
   LEGENDSIZE=" width, height, autoextend"
   SWATCHSIZE="width, height"
   BACKGROUNDCOLOR = "R,G,B"
   TRANSCOLOR="R,G,B"
   CELLSPACING="int value"
   REVERSEORDER= "TRUE|FALSE"
   VALUEMAPSPLIT = "cansplit, split text, column no"
   GENERATEHTML="TRUE|FALSE"  >
```

Extended syntax

To specify layer visibility using subtags instead of LAYERVISIBLELIST and LAYERINVISIBLELIST attributes, use one or more CF_ARCIMS_LAYER tags inside CF_ARCIMS.

```
   <CF_ARCIMS_LAYER  LayerID="Layer_ID"  Visible=[true|false] >
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
BACKGROUNDCOLOR	N	Color	255, 255, 255	0,0,0–255,255,255	Background color of legend.
CELLSPACING	N	Integer	3	N/A	Number of pixels between entries.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
FONT	N	String	Arial (font name)	N/A	Specification of font to draw text in the legend. String is comprised of font name, font size, font style, font color. Font name must exist on Spatial Server. Font size is in points. Font styles are Regular, Bold, Italic, Underline, or Outline. Font color is RGB specification of the text color.
GENERATEHTML	N	Boolean	False	True, False	If true, HTML IMG tag is generated in place of CF_ARCIMS during run time. If false, all variables are set but no tag is generated.
LAYERINVISIBLE LIST	N	Integers	N/A	N/A	List of IDs for layers to turn on.
LAYERLISTORDER	N	Boolean	False	True, False	If true, layers are drawn in order listed in LAYERVISIBLELIST, and only layers specified in LAYERVISIBLELIST are used. If false, layers are drawn in theorder they appear in map file and all visible layers are used.
LAYERVISIBLE LIST	N	Integers	N/A	N/A	List of IDs for layers to turn off.
LEGENDSIZE	N	String	125,300,true	N/A	The width and height of legend image and autoextend value. Autoextend is a Boolean value for overriding the legend height if there are more layers to show than fit in the specified size. If autoextend is true, legend image may be higher than the specified height.
REVERSEORDER	N	Boolean	False	True, False	If true, layer order is reversed.
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to draw the legend for.
SWATCHSIZE	N	Double	18, 12	N/A	Width and height of the area containing the symbol.
TITLE	N	String	N/A	N/A	Title of the legend.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
TRANSCOLOR	N	Color	N/A	0,0,0–255,255,255	Image color to be transparent.
VALUEMAPLIST	N	String	False	N/A	Defines how to split value map layers between columns in format Boolean, string, integer. First value determines if values are split. Second value is the text at the bottom of the column that is wrapped into the new column. Last value specifies how many columns layers can be split into.

Subtags

Name	Required	Occurrences	Notes
CF_ARCIMS_LAYER	N	Many	Used to specify layers used in the legend. There should be one CF_ARCIMS_LAYER for every layer that needs to be turned on or off. If LAYERLISTORDER attribute is false, only layers with visibility different from the default need to be specified. If LAYERLISTORDER is true, only layers specified with CF_ARCIMS_LAYER are extracted, so all layers that need to be extracted should be specified, regardless of the default visibility. Replaces LAYERVISIBLELIST and LAYERINVISIBLELIST.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_IMAGEPATH	Y	String	Path of image generated by the Spatial Server as seen by the Spatial Server.
OUT_IMAGEURL	Y	String	URL of image generated by the Spatial Server.

Variable Syntax	Always Generated	Variable Type	Usage
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

None

Notes

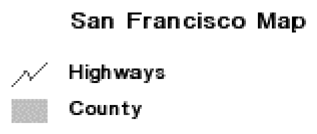
- A LEGEND request creates an image that contains the legend for the MapService. It is usually used in conjunction with a GENERATEMAP action—GENERATEMAP creates the map image and LEGEND creates the legend.
- A legend created by the request is an image, just like the map image, and can be used in the same way. If the GENERATEHTML attribute is set to 'true', an IMG HTML tag is generated for the user. For more control, a developer can set GENERATEHTML to false, and use an OUT_IMAGEURL CF variable that contains the URL of the generated legend image.
- The legend created has a section for every active layer and an entry for every class used. The user has control over class descriptions using the 'label' attribute of EXACT or RANGE tags in the map configuration file.
- LEGEND is scale sensitive. Only layers that are visible at the current scale appear in the legend. However, since it is not possible to set the map scale using a CF_ARCIMS LEGEND action, only layers visible at the default scale appear in the legend. To set a scale other than the default, the developer must use a CF_ARCIMS REQUEST action with the ArcXML LEGEND tag.
- More customized legends can be created using the REQUEST action with ArcXML LEGEND tag. Using this approach a developer can define on-the-fly custom layers and renderers the same way he would do it for GET_IMAGE request. A developer can change the scale to keep the legend in sync with the map. For more information on how to use the ArcXML LEGEND tag, see the *ArcXML Programmer's Reference Guide*.

Examples

1) Creating a simple legend based on the MapService defaults.

```
<cf_arcims  action=”legend”
   ServiceName=”SanFrancisco”
   ServerName=”bear”
   ServerPort=”5300"
   Title=”San Francisco Map”
   LegendSize=”200,400,true”
   generateHTML=”true”/>
```

This generates the following image:



2) To turn the County layer off and leave only Highways, you need to know the ID of the County layer. In this MapService, the County layer has ID 0. To turn it off, it can either be included at LAYERINVISIBLELIST or be specified using CF_ARCIMS_LAYER.

```
<cf_arcims  action=”legend”
   ServiceName=”SanFrancisco”
   ServerName=”bear”
   ServerPort=”5300"
   Title=”San Francisco Map”
   LegendSize=”200,400,true”
   layerinvisiblelist=”0"
   generateHTML=”true” />
```

3) This example shows a page that displays both the map and appropriate legend. This works only at the default map extent and scale. If the map was zoomed in (e.g., extent is other than the default) layers on the map and on the legend do not match. To resolve this issue, the developer must use CF_ARCIMS REQUEST.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
   <title>Untitled</title>
</head>

<body>

<table>
<tr>
   <td>
   <cf_arcims action="generatemap"
          ServiceName="SanFrancisco"
          ServerName="bear"
          ServerPort="5300"
         envelope="-122.52809,37.634806,-122.310880,37.840374"
          imagesize="400,400"
          generateHTML="true" />
   </td>
   <td>
   <cf_arcims  action="legend"
          ServiceName="SanFrancisco"
          ServerName="bear"
          ServerPort="5300"
          Title="San Francisco Map"
          LegendSize="200,400,true"
          generateHTML="true" />
   </td>
</tr>
</table>

</body>
</html>
```

QUERY

Purpose

Performs an attribute, spatial, or combined ArcIMS query and returns feature attributes and geometry description.

Syntax

SQL query and spatial filter can be specified using either attributes or subtags. The advantage of using subtags is that single quotes can be used; however, ColdFusion special characters such as ‘<’ and ‘>’ still need to be escaped.

Basic syntax

```
<CF_ARCIMS ACTION=”QUERY”
   SERVERNAME=”server name”
   SERVERPORT=”server port”
   SERVICENAME=”service name”
   RETURNGEOMETRY=”TRUE|FALSE”
   RETURNENVELOPE=”TRUE|FALSE”
   RETURNCOMPACT=”TRUE|FALSE”
   LAYERID=”id”
   LAYERDISPLAYFIELDS=”field1 field2…”
   FEATURELIMIT=”number”
   RELATION=”relation”
   ENVELOPE=”minx, miny, maxx, maxy”
   Multipoint = “x1,y1,x2,y2,x3,y3,x4,y4”
   PolyLine=”x1,y1,x2,y2,x3,y3 ; x4,y4,x5,y5,x6,y6”
   Polygon=”x1,y1,x2,y2,x3,y3,x4,y4 ; x5,y5,x6,y6,x7,y7 : x8,y8,x9,y9,x10,y10”
   WHERE =”SQL expression”
   BUFFER=”distance, unit”
   BUFFERTARGETLAYERID = “id”
   JOINTABLES=”table”
   GENERATEHTML=”TRUE|FALSE” >
```

Extended syntax

The CF_ARCIMS_SQL subtag can be used instead of the WHERE attribute.

```
<CF_ARCIMS_SQL>
     [User SQL Statement]
</CF_ARCIMS_SQL>
```

Subtags also can be used instead of the MULTIPOINT, POLYLINE, and POLYGON attributes.

```
<CF_ARCIMS_POLYLINE>
   <path><Point x=”x1” y=”y1”/><Point x=”x2” y=”y2”/><point x=”x3” y=”y3”></path>
   <path><Point x=”x4” y=”y4”/> <Point x=”x5” y=”y5”/></path>
</CF_ARCIMS_POLYLINE>

<CF_ARCIMS_POLYGON>
   <Ring><Point x=”x1” y=”y1”/><Point x=”x2” y=”y2”/><Point x=”x3” y=”y3”/></Ring>
   <Ring><Point x=”x4” y=”y4”/><Point x=”x5” y=”y5”/><Point x=”x6” y=”y6”/><Point
   x=”x7” y=”y7”/>
<Hole>
   <Point x=”x8” y=”y8”/><Point x=”x9” y=”y9”/>
</Hole>
</Ring>
</CF_ARCIMS_POLYGON>

<CF_ARCIMS_MULTIPOINT>
   <Point x=”x1” y=”y1”/>
   <Point x=”x2” y=”y2”/>
   <Point x=”x3” y=”y3”/>
</CF_ARCIMS_MULTIPOINT>
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
BUFFER	N	String	N/A	N/A	If specified, a buffer is created around selected features on LAYERID and the buffer is applied to select features on BUFFERTARGETLAYERID. Features from BUFFERTARGETLAYERID are returned. Format of attribute value is distance, units. Distance is buffer distance. Units are distance units. Valid keywords for units are DECIMAL_DEGREES, MILES, FEET, KILOMETERS, and METERS (case sensitive. If BUFFER attribute is used, BUFFERTARGETLAYERID is also required.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
BUFFERTARGET LAYERID	N	String	N/A	N/A	Specifies layer containing the features that are selected by the buffer specified in BUFFER attribute. If BUFFER and BUFFERTARGETLAYERID are specified, features from BUFFERTARGETLAYERID are returned and not those from LAYERID. This attribute is always used with BUFFER.
ENVELOPE	N	String	N/A	N/A	Envelope used as a spatial filter. Features inside the envelope are selected. Coordinates are in map coordinate system.
FEATURELIMIT	N	Integer	All features	N/A	Maximum number of features returned.
GENERATEHTML	N	Boolean	False	True, False	If true, HTML table is generated in place of CF_ARCIMS tag during the run time; if false, all variables are set but no table is generated.
JOINTABLES	N	Strings	N/A	N/A	List of tables that take part in a query. Join statement must be specified in the WHERE attribute and fieldsto retrieve from join tables must be listed in LAYERDISPLAYFIELDS. Depending on the data source type, a fully qualified table name (owner.table) may be required. Base layer table does not need to be specified. Tables can be joined only if the layer uses an ArcSDE data source. DBase tables (.dbf) can't be joined.
LAYERDISPLAY TABLES	N	Strings	N/A	#ALL#	List of layer attributes to return from the Spatial Server. To retrieve all fields, use #ALL#. Depending on the data source type and the query structure, a fully qualified column name (user.table.column) may be required.
LAYERID	Y	String	N/A	N/A	ID of the layer to query.
MULTIPOINT	N	String	N/A	N/A	Coordinates (x1,y1,x2,y2,...) of points used as a spatial filter.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
POLYGON	N	String	N/A	N/A	Polygon to use as a spatial filter. Polygon is defined as a set of one or more rings. Each ring contains points. Multiple rings are separated by ';'. A ring can contain holes. Holes are defined inside a ring by adding the points representing them after defining the points representing the ring, but separating them by ':'. Coordinates are in map units (x11,y11,x12,y12; x21,y21,x22,y22;...).
POLYLINE	N	String	N/A	N/A	Coordinates of the polyline used as a spatial filter. Polyline is defined as one or more paths. A path consists of points. Multiple paths in a polyline are separated by ';'. Coordinates are in mapunits (x11,y11,x12,y12;x21, y21,x22,y22;...).
RELATION	N	String	area_ intersection	area_ intersection envelope_ intersection	The type of spatial relation between the spatial filter and the query features.
RETURNCOMPACT	N	Boolean	False	True, False	If true, ArcXML geometry is returned in compact form. It should be used only if RETURNGEOMETRY="True".
RETURNENVELOPE	N	Boolean	False	True, False	If true, feature envelope is returned with other attributes. If using ArcSDE layers, spatial column name must be specified in LAYERDISPLAYFIELDS. Envelope is in four new OUT_QUERYTABLE fields: MINX, MINY, MAXX, and MAXY.
RETURNGEOMETRY	N	Boolean	False	True, False	If true, ArcXML representation of feature geometry is returned with other attributes. If using ArcSDE layers, spatial column name must be specified in LAYERDISPLAYFIELDS. Geometry map file is in the OUT_QUERYTABLE column AXLGEOMETRY.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to be drawn.
WHERE	N	String	N/A	N/A	Contains the ‘where’ clause of the SQL statement. Used instead of the WHERE features.

Subtags

Name	Required	Occurrences	Notes
CF_ARCIMS_MULTIPOINT	N	One	Defines points to be used as a spatial filter. Multiple points can be specified using the POINT tag inside one CF_ARCIMS_MULTIPOINT tag.
CF_ARCIMS_POLYGON	N	One	Defines polygons to be used as a spatial filter. Polygons are specified using RING and HOLE sub tags.
CF_ARCIMS_POLYLINE	N	One	Defines polylines to be used as a spatial filter. Multiple lines can be specified using the PATH tag inside CF_ARCIMS_POLYLINE. Each PATH tag can have two or more POINT tags.
CF_ARCIMS_SQL	N	One	Contains the ‘where’ clause of the SQL statement. Used instead of the WHERE attribute to select features.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_QUERYTABLE	N	CF Query	CF query containing records for all selected features. Query fields are specified by LAYERDISPLAYFIELDS or by the MapService if #ALL# is used. This can be used as a standard CF query in <cfoutput>, <cfloop>, and all other CF tags and functions that expect a CF query as an argument. If RETURNGEOMETRY is true, this table contains the ArcXML representation of the geometry of the features in the AXLGEOMETRY column. If RETURNENVELOPE is true, this table contains MINX, MINY, MAXX, and MAXY with the envelope of every returned feature. If there are no features selected, the query is not defined, so always check.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

- Buffer processing is somewhat limited compared to ArcIMS Spatial Server capabilities. The buffer can be used to select features from another layer, but the buffer itself cannot be retrieved.

Notes

- This tag is used to perform spatial, attribute, or combined queries on layers. The result is a ColdFusion query containing attributes and, if requested, the geometry of the selected features, but no image is generated. To highlight a selected feature on the map image, the QUERY request must be used together with a GENERATEIMAGE request.
- Queries can be made to either ArcIMS Image or Feature MapServices.
- There are two ways to use related tables in the query. Related tables and a join statement can be specified in the layer definition in the MapService. In this case, columns from the related table can be used in the same way as columns from the original tables. Related tables and a join statement can be specified during

the query in the JOINTABLES and WHERE attributes. JOINTABLES specifies which tables to join (base layer table is assumed and does not need to be specified), and WHERE specifies the join criteria using standard SQL syntax. Joins must be one-to-one or many-to-one. All tables used in the query must be in the same ArcIMS workspace. It is not possible to join, for example, ArcSDE layers from two different ArcSDE instances. When using joins, column names in LAYERDISPLAYFIELDS should be fully qualified (table_name.column_name). Using this tag, it is only possible to use join tables if the layer data source is ArcSDE. To use join tables with layers with a shapefile data source, use the REQUEST tag.

- GENERATEHTML= “true” is used primarily for testing, since it does not provide a method to format the output. Applications in a production environment would probably use CF functions and tags to loop through OUT_QUERYTABLE and format the result.
- If geometry is requested, it is returned in the AXLGEOMETRY field in ArcXML format. For map file geometry specifications, see the *ArcXML Programmer's Reference Guide*. If the source of the requested layer is ArcSDE, to retrieve the geometry or envelope the ArcSDE spatial column must be specified in LAYERDISPLAYFIELDS.

Examples

1) A simple attribute query:

```
<cf_arcims  action=”query”
   ServiceName=”SanFrancisco”
   ServerName=”bear”
   ServerPort=”5300”
   LayerID=”2"
   layerdisplayfields=”##ALL##”
   featurelimit=”100”
   where=”HWYNAME=’MARKET ST’”
   generatehtml=”true”/>
```

This query returns records for all segments of Market St with all attributes, but no geometry or envelope. Since GENERATEHTML is true, an HTML table is automatically generated.

2) A similar query uses the CF_ARCIMS_SQL subtag and contains some post processing of the output query table.

   Instead of generating an HTML table, this example loops through a result set, sums the length field, and displays the total sum.

```
<cf_arcims  action=”query”
   ServiceName=”SanFrancisco”
   ServerName=”bear”
   ServerPort=”5300”
   LayerID=”2”
   layerdisplayfields=”##ALL##”
   featurelimit=”100”
   generatehtml=”false”>
   <CF_ARCIMS_SQL>
      HWYNAME=’MARKET ST’
   </CF_ARCIMS_SQL>
</cf_arcims>
<cfset len=0>
<cfloop query=”out_querytable”>
   <cfset len = len + out_querytable.length>
</cfloop>
<cfoutput>
   Total length of the Market Street is #len# miles.
</cfoutput>
```

3) A combination of attribute and spatial query. The goal is to select census blocks in the notheast part of the town with a median age less than 35. Notice the escape characters &lt; to create the less than sign (<).

```
<cf_arcims  action=”query”
   ServiceName=”Demog”
   ServerName=”bear”
   ServerPort=”5300"
   LayerID=”3"
   layerdisplayfields=”##ALL##”
   relation=”area_intersection”
   generatehtml=”true” >

   <CF_ARCIMS_SQL>
      MED_AGE &lt; 35
   </CF_ARCIMS_SQL>

   <CF_ARCIMS_POLYGON>
      <RING>
        <POINT x=”-122.45" y=”37.82" /> <POINT x=”-122.44" y=”37.76" />
        <POINT x=”-122.37" y=”37.77" /> <POINT x=”-122.38" y=”37.83" />
      </RING>
   </CF_ARCIMS_POLYGON>

</cf_arcims>
```

IDENTIFY

Purpose

Identifies the feature at a given point and retrieves the feature attributes.

Syntax

Basic syntax

```
<CF_ARCIMS ACTION="IDENTIFY"
   SERVICENAME="service name"
   SERVERNAME="server name"
   SERVERPORT="server port"
   ATTRIBUTES="attributes"
   LAYERID="id"
   ENVELOPE="minx,miny,maxx,maxy"
   IMAGESIZE="width,height"
   POSX="x"
   POSY="y"
   TOLERANCE="tolerance"
   GENERATEHTML="TRUE|FALSE">
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
ATTRIBUTES	N	Strings	#ALL#	N/A	List of feature attributes to be retrieved from the Spatial Server. To retrieve all attributes use '#ALL#'.
ENVELOPE	Y	Doubles	N/A	N/A	Map extent of map image in map units the request refers to as a comma-separated list (minX, minY, maxX, maxY).
GENERATEHTML	N	Boolean	False	True, False	If true, HTML IMG table is generated in place of CF_ARCIMS tag during the run time; if false, all variables are set but no table is generated.
IMAGESIZE	Y	Numbers	N/A	N/A	Width and height of map image in pixels.
LAYERID	Y	String	N/A	N/A	ID of layer to query.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
POSX	Y	Number	N/A	N/A	X-coordinate in the image coordinate system (pixels) of the point used for identification.
POSY	Y	Number	N/A	N/A	Y-coordinate in the image coordinate system (pixels) of the point used for identification.
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to be drawn.
TOLERANCE	N	Integer	2	N/A	Search distance around click location in pixels.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_QUERYTABLE	N	CF Query	CF query containing records for all selected features.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

- IDENTIFY identifies features from the specified layer, so it fits well into applications that use the 'active layer' concept. To identify features from more than one layer, use several IDENTIFY tags in the loop that goes through layers.

Notes

- IDENTIFY always works in conjunction with the map image generated using GENERATEMAP. Standard procedure is to use GENERATEMAP to create an INPUT type image or an IMG HTML tag, ask a user to click on the map, capture click coordinates, submit an IDENTIFY request with the click coordinates, and use the OUT_QUERYTABLE query to present the user with the result. The easiest way to capture click coordinates is to use an INPUT type image HTML table in the form and specify a CFM page with IDENTIFY as a form action. Coordinates of the mouse click are in the FORM.map_image.X and FORM.map_image.Y variables in the called page (map_image is the name of the image input field containing the map image in the calling page).
- Depending on the value of the TOLERANCE attribute, there will be 0, 1, or more than one feature in the result set. If there are no selected features, OUT_RESULTQUERY is not defined. Always check if a query is defined before trying to access the values.
- GENERATEHTML="true" is used primarily for testing, since it does not provide a method of formatting the output. Applications in a production environment would probably use CF functions and tags to loop through OUT_QUERYTABLE and format the result.

Examples

1) This request returns information about a ZIP Code polygon (layer ID 1) in the center of the map image:

```
<CF_ARCIMS ACTION=”IDENTIFY”
   SERVICENAME=”SanFrancisco”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   ATTRIBUTES=”##ALL##”
   LAYERID=”1"
   ENVELOPE=”-122.46,37.76,-122.42,37.79"
   IMAGESIZE=”400,400"
   POSX=”200"
   POSY=”200"
   TOLERANCE=”2"
   GENERATEHTML=”true”>
```

Since GENERATEHTML=”true”, the tag generates an HTML table.

2) The following CFM page shows the map image inside the HTML form then, as a form action, runs Identify. Identify results are shown in a formatted table.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>

<html>
<head>
   <title>Identify Example</title>
</head>

<body>

<cfif isDefined(“Form.map_image.x”)>
   <!—
   This is executed when page is invoked second time, from the form.
   —>
   <CF_ARCIMS ACTION=”IDENTIFY”
      SERVICENAME=”SanFrancisco”
      SERVERNAME=”bear”
      SERVERPORT=”5300"
      ATTRIBUTES=”##ALL##”
      LAYERID=”5"
      ENVELOPE=”#Form.Extent#”
      IMAGESIZE=”400,400"
      POSX=”#Form.Map_Image.X#”
      POSY=”#Form.Map_Image.Y#”
      TOLERANCE=”2"
      GENERATEHTML=”false”>

   <cfif isDefined(“out_querytable”)>
      <!— Result query exist, so user selected the theater —>
      <cfoutput query=”OUT_QUERYTABLE”>
        <table>       <tr>
              <td><b>Name:</b></td><td>#out_querytable.name#</td>
        </tr><tr>
              <td><b>Address:</b></td><td>#out_querytable.address#,
              #out_querytable.city#</td>
        </tr><tr>
              <td><b>Phone:</b></td>
              <td>#out_querytable.phone#</td>
        </tr> </table><br><br>
      </cfoutput>
   <cfelse>
```

(continued on next page)

```
      <!— Result query does not exist, user did not select the theater —>
      No theaters found. Theatres are small red triangles. Click ‘Back’ button and
      try again.
    </cfif>

  <cfelse>
    <!—
    This is executed first time page is invoked.
    —>
    <CF_ARCIMS ACTION=”GENERATEMAP”
      SERVERNAME=”bear”
      SERVERPORT=”5300"
      SERVICENAME=”SanFrancisco”
      ENVELOPE=”-122.46,37.76,-122.42,37.79"
      IMAGESIZE=”400,400"
      GENERATEHTML=”false”/>

    <b>Click on the map to identify the theater:</b><br>
    <cfoutput>
    <form action=”id2.cfm” method=”POST”>
      <input type=”Image” name=”Map_Image” src=”#OUT_IMAGEURL#” width=”400"
      height=”400">
      <input type=”Hidden” name=”Extent”
      value=”#OUT_MINX#,#OUT_MINY#,#OUT_MAXX#,#OUT_MAXY#”>
    </form>
    </cfoutput>

  </cfif>

  </body>
  </html>
```

GETMAPSERVICES

Purpose

Gets a list of the MapServices defined on the Spatial Server and basic information about each MapService.

Syntax

```
<CF_ARCIMS ACTION=”GETMAPSERVICES”
SERVERNAME=”server name”
SERVERPORT=”server port”
GENERATEHTML=”TRUE|FALSE” >
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
GENERATEHTML	N	Boolean	False	True, False	If true, HTML IMG table is generated in place of CF_ARCIMS tag during the run time; if false, all variables are set but no table is generated.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_SERVICETABLE	Y	CF Query	CF query containing records for all MapServices currently running on the Spatial Server. The fields in the query are described in the notes below. This query can be used as a standard CF query in <cfoutput>, <cfloop>, and all other CF tags and functions that expect a CF Query as an argument.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.

Variable Syntax	Always Generated	Variable Type	Usage
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

None.

Notes

- This request is used to retrieve a list of the MapServices currently running on the specified Spatial Server and basic MapService information. To get more detailed information about a MapService, use a GETSERVICEINFO request.
- Fields in OUT_SERVICESTABLE are:

NAME	Name of the MapService
SERVICEGROUP	Virtual server associated with the MapService
ACCESS	Service type access (PUBLIC or PRIVATE)
TYPE	Service type (ImageServer, FeatureServer, GeocodeServer)
DESC	Description
GROUP	Security group (not used)
STATUS	Current MapService status (ENABLED or DISABLED)
AXL_PATH	Name of the map file defining the MapService
URL	URL of the image generated by the Image Server
PATH	Path of the image generated by the Image Server as the Spatial Server sees it
IMAGE_TYPE	Type of image generated by the Spatial Server (GIF, PNG, PNG8, JPEG)

Examples

1) The basic usage of the request:

```
<cf_arcims  action=”getmapservices”
   ServerName=”bear”
   ServerPort=”5300”
   Generatehtml=”True”/>
```

Since GENERATEHTML is set to true, an HTML table is created.

2) How to use GETMAPSERVICES to give the user a list of MapServices to select from:

```
<cf_arcims  action=”getmapservices”
   ServerName=”bear”
   ServerPort=”5300”
   Generatehtml=”false”/>

Please select the MapService:<br><br>

<form action=”” method=”POST”>
<select name=”service”>
<cfoutput query=”out_servicestable”>
   <option value=”#out_servicestable.name#”>#out_servicestable.name#</option>
</cfoutput>
</select>
</form>
```

Since GENERATEHTML is set to false, no table is automatically generated. The resulting out_servicestable query is used inside the <cfoutput> tag to generate a set of <option> tags for the list in the form.

GETSERVICEINFO

Purpose

Gets detailed information about a MapService.

Syntax

```
<CF_ARCIMS ACTION="GETSERVICEINFO"
   SERVICENAME="service name"
   SERVERNAME="server name"
   SERVERPORT="server port"
   GENERATEHTML="TRUE|FALSE" >
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
GENERATE HTML	N	Boolean	False	True, False	If true, HTML tables are generated in place of the CF_ARCIMS tag during run time; if false, all variables are set but no table is generated.
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to be drawn.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_GEOCODE	N	CF Query	CF query generated only if MapService type isSTYLES 'Geocode'. It contains a list of the MapService layers with geocoding extensions. The fields in the query are described in the notes below.
OUT_*LayerName*_ FIELDTABLE	Y	CF Query	Set of CF queries containing information about fields in each layer. The fields in the query are described in the notes below.

Variable Syntax	Always Generated	Variable Type	Usage
OUT_LAYERTABLE	Y	CF Query	CF query containing information about all layers defined in the MapService. The fields in the query are described in the notes below. This query can be used as a standard CF query in <cfoutput>, <cfloop>, and all other CF tags and functions that expect a CF query as an argument.
OUT_MAPENVELOPE TABLE	Y	CF Query	CF query containing default map extent of the MapService. The fields in the query are described in the notes below. This query can be used as a standard CF query in <cfoutput>, <cfloop>, and all other CF tags and functions that expect a CF query as an argument.
OUT_*StyleName*_ INPUTS	N	CF Query	Set of CF queries containing a description of required inputs used for the style specified in the name of the query object. There is one query for every geocoding style used in the MapService. The fields in the query are described in the notes below.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

None.

Notes

- A GETSERVICEINFO request can be used to retrieve detailed information about a MapService and layers in the MapService up to field definition for each layer. Information is returned in a set of CF queries.

- OUT_MAPENVELOPETABLE has just one record and four fields representing the coordinates of the default MapService's map extent. The extent is the default extent set in the Map configuration file (.axl). Fields are:

MINX	X-coordinate of the lower-right corner
MINY	Y-coordinate of the lower-right corner
MAXX	X-coordinate of the upper-right corner
MAXY	Y-coordinate of the upper-right corner

- OUT_LAYERTABLE contains a record for every layer in the MapService regardless of visibility. Fields are:

NAME	Layer name
LAYERTYPE	Type of the layer (featureclass or image)
FEATURETYPE	Feature type of the layer (point, line or polygon) if layer is based on vector data (ArcSDE or shapefiles)
VISIBLE	Layer visibility (true or false)
MAXSCALE	Maximum scale at which layer displays
MINSCALE	Minimum scale at which layer displays
MINX	X-coordinate of the lower-right corner
MINY	Y-coordinate of the lower-right corner
MAXX	X-coordinate of the upper-right corner
MAXY	Y-coordinate of the upper-right corner
ID	Layer ID

- OUT_*LayerName*_FIELDTABLE contains field definitions for every layer. There is one OUT_*LayerName*_FIELDTABLE for every layer where *LayerName* is the name of the layer or ID of the layer if the name is not defined. If the name or ID contains dots ('.'), the dots are replaced with the string '_ESRIDOT_'. If the name or ID contains spaces, the spaces are replaced with underscores ('_'). OUT_*LayerName*_FIELDTABLE has one record for every field in the layer. OUT_*LayerName*_FIELDTABLE fields are:

NAME	Name of the field
TYPE	Field type
SIZE	Field size
PRECISION	Field precision

- An OUT_GEOCODE_STYLES query is created only if the MapService has geocode properties set. It contains a record for every layer that is prepared for geocoding and the geocoding style associated with it. Fields are:

LAYERNAME	Layer name
LAYERID	Layer ID
STYLENAME	Name of the geocoding style associated with the layer

- OUT_*StyleName*_INPUTS contains definitions of the input data required for every style type. There is an OUT_*StyleName*_INPUTS defined for every geocoding style used in the MapService. There will be one record for every required parameter for a particular style. Fields are:

ID	Parameter ID
LABEL	Parameter name
WIDTH	Parameter width
TYPE	Parameter type
DESCRIPTION	Parameter description

Examples

1) The basic usage of the tag:

```
<cf_arcims action="getserviceinfo"
   ServiceName="SanFrancisco"
   ServerName="bear"
   ServerPort="5300"
   generateHTML="true" />
```

GENERATEHTML is true, so HTML tables are created.

2) This example puts layer names in a table with a green background for visible layers and a red one for invisible layers. It demonstrates the use of OUT_LAYERTABLE:

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>

<html>
<head>
   <title>GETSERVICEINFO Example</title>
</head>

<body text=”white”>

<cf_arcims  action=”getserviceinfo”
   ServiceName=”SanFrancisco”
   ServerName=”bear”
   ServerPort=”5300"
   generateHTML=”false” />

<table border=0 >
<tr bgcolor=”Teal”>
   <td><b>ID</b></td>
   <td><b>Name</b></td>
   <td><b>Layer Type</b></td>
   <td><b>Feature Type</b></td>
</tr>
<cfoutput query=”out_layertable”>
<cfif #out_layertable.visible# eq “true”>
   <cfset bgcol=”Green”>
   <cfelse>
   <cfset bgcol=”Maroon”>
   </cfif>
<tr bgcolor=”#bgcol#”>
   <td>#out_layertable.ID#</td>
   <td>#out_layertable.NAME#</td>
   <td>#out_layertable.LAYERTYPE#</td>
   <td>#out_layertable.FEATURETYPE#</td>
</tr>
</cfoutput>
</table>

</body>
</html>
```

GEOCODE

Purpose

Performs address matching and geocoding.

Syntax

```
<CF_ARCIMS ACTION = "GEOCODE"
   SERVICENAME ="service name"
   SERVERNAME="server name"
   SERVERPORT="server port"
   STREET="street address OR cross street #1"
   LAYERID="1"
   CrossStreet="cross street #2"
   ZONE="zip code OR any other zoning value"
   KeyField="value"
   MAXCANDIDATES="int value"
   MINSCORE="min acceptable score"
   PINPOINT="TRUE|FALSE"
   GENERATEHTML="TRUE|FALSE"  >
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
CROSSSTREET	N	String	N/A	N/A	If address is given as a street intersection, this field contains the name of the second intersection street.
GENERATEHTML	N	Boolean	False	True, False	If true, HTML IMG table is generated in place of CF_ARCIMS during run-time. If false, all variables are set but no table is generated.
KEYFIELD	N	String	N/A	True, False	Used for 'SingleField' geocoding style, contains the value for a specific field.
LAYERID	Y	String	N/A	N/A	ID of layer to query.
MAXCANDIDATES	N	Integer	5	N/A	Defines number of candidates returned to client when a specified address cannot be resolved with a matching score of 100.
MINSCORE	N	Integer	60	1–100	Minimum acceptable score of returned candidates.
PINPOINT	N	Boolean	True	True, False	If true, returns addresses' point locations.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
SERVERNAME	Y	String	N/A	N/A	Name of the server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
STREET	N	String	N/A	N/A	Depending on geocoding style, this can contain an address or first intersection street.
ZONE	N	Boolean	N/A	True, False	If an address style with a zone is used, this can contain zone information. Zone is usually a ZIP Code.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_GEOCODE RESULTS	N	CF Query	CF Query containing geocoding results. It hasone record for each returned candidate.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	N/A Where the Spatial Server returns error messages.

Restrictions

- Before it can be used for geocoding, an layer must be prepared for a particular geocoding style.

Notes

- Address geocoding in ArcIMS is a process that creates points based on address data. During geocoding, ArcIMS reads these addresses and locates them on a street network layer. Resulting points can be displayed with other layers in an acetate layer using a GENERATEMAP request. ArcIMS returns a result set represented as a CF Query in ColdFusion with the address and the X,Y coordinates of the successfully matched records.
- Events can be geocoded using standard addresses and also using street intersection, ZIP Code, or place name. Prior to geocoding, the MapService must have a street network or Zip Code layer with the data prepared for a particular geocoding style. For more information on setting up geocoding in your MapService see *Using ArcIMS,* Chapter 3, 'Setting geocoding properties'.

  Typical geocoding procedure:

  1. Setup your Map configuration file (.axl) in ArcIMS Author for geocoding. For more information on setting up geocoding in your Map configuration file see *Using ArcIMS,* Chapter 3, 'Setting geocoding properties'.
  2. Create an ArcIMS MapService from the map file.
  4. Send the geocoding request using the GEOCODE tag.
  5. Process the results. Results are returned in a CF Query and usually contain the coordinates of the matched locations. These locations can be drawn as points on the acetate layer in a GENERATEMAP request.
- The address description required for geocoding varies depending on the geocoding style. The geocoding style is determined when an ArcIMS layer is prepared for the geocoding. To find the geocoding style on a particular layer, use the GETSERVICEINFO request. If the layer in the request is enabled for geocoding, GETSERVICEINFO creates an OUT_GEOCODE_STYLES CF Query listing of the geocoding style of the layer.
- Geocoding styles and required attributes follow. For more information on Geocoding styles, see *Using ArcIMS*, Chapter 3, 'Geocoding styles'.

USAddress	Style 'US streets' requires STREET; optional CROSSSTREET. STREET contains the event address or just a street name if CROSSSTREET is specified.
USAddressZ	Style 'US streets with zone' requires STREET; optional CROSSSTREET and ZONE. STREET contains the event address or just a street name if CROSSSTREET is specified. ZONE is the ZIP Code or any other zone specifier.
SingleField	Style 'Single field' requires a KEYFIELD. This address style is used for geocoding based on the value of a single field in the layer's attribute table.
USSingleHouse	Style 'US single house' requires STREET; optional CROSSSTREET.
USSingleHouseZ	Style 'US single house with zone' requires STREET; optional CROSSSTREET and ZONE.
USSingleRange	Style 'US single range' requires STREET; optional CROSSSTREET. STREET contains the event address, or just a street name if CROSSSTREET is specified.

USSingleRangeZ	Style 'US single range with zone' requires STREET; optional CROSSSTREET and ZONE. STREET contains the event address, or just a street name if CROSSSTREET is specified.
Zip4	Style 'Zip+4' requires a value with 4 digits representing the ZIP Code.
Zip4Range	Style 'Zip+4 range requires a value with 4 digits representing ZIP code. This request is similar to Zip4. The only difference is that more than one ZIP Code can matched to one reference point, e.g., the point in the reference layer can represent more than one ZIP code.
Zip5	Style 'Zip 5-digit' requires a value with 5 digits representing ZIP Code. The 5-Digit ZIP address style can be used for geocoding 5-digit ZIP code addresses.
Zip5Range	Style 'Zip 5-digit range' requires a value with 5 digits representing ZIP Code. This request is similar to Zip5. The only difference is that more than one ZIP Code can match to one reference point, e.g., a point in the reference layer can represent more than one ZIP Code.

- OUT_GEOCODERESULTS CF Query contains address and (if PINPOINT is true) location of matched events. Fields in the query are:

Score	Matching score for a particular candidate (1–100)
AddressFound	Address for a candidate as found by ArcIMS
PointX	X-coordinate of the point
PointY	Y-coordinate of the point

Examples

1) Simple geocoding on a street network layer.

```
<cf_arcims  action="geocode"
   ServiceName="geocoding"
   ServerName="bear"
   ServerPort="5300"
   Street="150 N. Palm Av."
   LayerID="0"
   MaxCandidates="10"
   MinScore="80"
   PinPoint="true"
   Zone="0"
   GenerateHTML="true"/>
```

Since MinScore is 80, only candidates greater than or equal to that score are returned. If MinScore is set to a lower number, more candidates can be reviewed.

2) Geocoding a street intersection. Resulting point coordinates are then used to draw the point in the map image.

   In this example, the REQUEST action is used instead of GENERATEMAP to create a map image. Using GENERATEMAP, which only supports acetate objects in pixel coordinates, and GEOCODE, which returns the point in database coordinates, you are required convert them to image coordinates. Converting a database to image coordinates is not a problem if the exact extent of the image generated is known. However, since the aspect ratio of image size and requested extent can differ, the extent of the map image generated by the Spatial Server can be different from the one requested. You must either ensure that the request extent always has the same aspect ratio as image, or send two GENERATEMAP requests with the first one designed to get the extent of the database coordinates. As an alternative, you can use the REQUEST action. The REQUEST action supports an acetate layer point in database coordinates, is probably the simplest and most flexible.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>

<html>
<head>
   <title>Geocoding Example 2</title>
</head>

<body>

<cf_arcims  action=”geocode”
   ServiceName=”geocoding”
   ServerName=”bear”
   ServerPort=”5300”
   Street=”W. C St.”
   CrossStreet=”N. Euclid”
   LayerID=”0”
   MaxCandidates=”10”
   MinScore=”30”
   PinPoint=”true”
   Zone=”0”
   GenerateHTML=”true”/>

<cf_arcims  action=”request”
   ServiceName=”geocoding”
   ServerName=”bear”
   ServerPort=”5300”
   ParseResponseAXL=”true”
   GenerateHTML=”true”>
```

(continued on the next page)

```
   <?xml version=”1.0”?>
    <ARCXML version=”1.0.1”>
    <REQUEST>
    <GET_IMAGE>
      <PROPERTIES>
        <IMAGESIZE width=”400" height=”400”/>
        <ENVELOPE minx=”6659644.408724”
                miny=”1841321.745850”
                maxx=”6675899.493225”
                maxy=”1851623.731328”/>
        </PROPERTIES>
        <LAYER type =”ACETATE” name=”POINTLAYER”>

          <cfloop query=”OUT_GEOCODERESULTS” >
          <cfoutput>
          <OBJECT units=”database”>
          <POINT coord=”#OUT_GEOCODERESULTS.PointX#,#OUT_GEOCODERESULTS.PointY#”>
                      <SIMPLEMARKERSYMBOL type=”circle” color = “255,0,0”
                              width = “10”/>
           </POINT>
           </OBJECT>
           </cfoutput>
           </cfloop>

        </LAYER>
      </GET_IMAGE>
      </REQUEST>
      </ARCXML>

   </CF_ARCIMS>

</body>
</html>
```

EXTRACT

Purpose

Extracts map data from requested database MapServices.

Syntax

Layer visibility can be specified using either CF_ARCIMS tag attributes or subtags.

Basic syntax

```
<CF_ARCIMS ACTION = “EXTRACT”
   SERVICENAME = “service name”
   SERVERNAME=”server name”
   SERVERPORT=”server port”
   ENVELOPE=“minx,miny,maxx,maxy”
   LAYERLISTORDER=”true | false”
   LAYERINVISIBLELIST=”layer1, layer2, layer3”
   LAYERVISIBLELIST=”layer4, layer5, layer6”
   OUTPUTURL=”URL for where the zip file will be available”
   OUTPUTPATH=”path for output of zip file”
   GENERATEHTML=”TRUE|FALSE”  >
```

Extended syntax

To specify layer visibility using subtags instead of LAYERVISIBLE LIST and LAYERINVISIBLELIST attributes, use one or more CF_ARCIMS_LAYER tags inside CF_ARCIMS.

```
<CF_ARCIMS_LAYER LayerID=”Layer_ID” Visible=[true|false] >
```

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
ENVELOPE	Y	Numbers	N/A	N/A	Extent of extract file to be generated in database units as a comma-separated list (MinX, MinY, MaxX, MaxY).
GENERATEHTML	N	Boolean	False	True, False	If true, HTML IMG tag is generated in place of CF_ARCIMS tag during the run time; if false, all variables are set but no tag is generated.
LAYERINVISIBLE LIST	N	String	N/A	N/A	List of IDs for layers to be turned on.
LAYERLISTORDER	N	Boolean	False	True, False	If true, only layers specified by LAYERINVISIBLELIST are used; if false, all visible layers are used.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
LAYERVISIBLE LIST	N	String	N/A	N/A	List of IDs for layers to be turned off.
OUTPUTPATH	N	String	From MapService	N/A	Path of location where export files are generated as seen by the Spatial Server.
OUTPUTURL	N	String	From MapService	N/A	URL of location where export files are generated as seen by the Spatial Server.
SERVERNAME	Y	String	N/A	N/A	Name of the server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to be extracted.

Subtags

Name	Required	Occurrences	Notes
CF_ARCIMS_LAYER	N	Many	Used to specify layers to be extracted. There should be one CF_ARCIMS_LAYER for every layer that needs to be turned on or off. If LAYERLISTORDER is false, only layers with visibility different from the default need to be specified. If LAYERLISTORDER is true, only layers specified with CF_ARCIMS_LAYER are extracted, so all layers that need to be extracted should be specified regardless of the default visibility. Replaces LAYERVISIBLELIST and LAYERINVISIBLELIST.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
OUT_FILEPATH	Y	String	Path of files generated by the Spatial Server as seen by the Spatial Server.
OUT_FILEURL	Y	String	URL of files generated by the Spatial Server.

Variable Syntax	Always Generated	Variable Type	Usage
OUT_MAXX	Y	Integer	Envelope (in map coordinates) of returned data.
OUT_MAXY	Y	Integer	Envelope (in map coordinates) of returned data.
OUT_MINX	Y	Integer	Envelope (in map coordinates) of returned data.
OUT_MINY	Y	Integer	Envelope (in map coordinates) of returned data.
IN_REQUESTAXL	Y	String	ArcXML code of the request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML code of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and places all relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

Restrictions

- Extract request does not support extracting the data from join tables, attribute query, or any other type of spatial filter other than envelope. To use those criteria, use a REQUEST action with the GET_EXTRACT tag.
- Extract virtual server must be manually set to PUBLIC access type. Default PRIVATE access type will not work. See notes below.

Notes

- EXTRACT request extracts the data from the ArcIMS data source (ArcSDE or shapefile) and creates a ZIP compressed archive file with the shapefiles containing requested features. The ZIP compressed archive file is placed in a directory specified by the OUTPUTPATH attribute, and OUTPUTURL plus the ZIP archive file name (generated by the Spatial Server) will be returned to the client. It is up to Webmaster to ensure that OUTPUTURL points to the OUTPUTPATH directory through an alias on the Web server. It is up to the application to retrieve the files for the client.
- By default, all currently visible layers are extracted. One set of files (.shp, .shx and .dbf) is created for each visible layer. To turn off visibility for a layer, include it in LAYERVISIBLELIST. A layer can be turned off if it is included in LAYERINVISIBLELIST. If LAYERLISTORDER is true, only layers in LAYERVISIBLELIST are extracted regardless of default visibility.

- The MapService used for extraction must use a EXTRACT virtual server. The Extract virtual server is initially defined with a PRIVATE access type. You are required to manually modified the access type to PUBLIC. To modify it, find the extract server configuration file at %AIMSHOME%\Server\etc\aimses.cfg on NT or $AIMSHOME/etc/aimses.cfg on UNIX, modify the line 'access = "PRIVATE"' to 'access = "PUBLIC"', and restart ArcIMS. Now the Extract virtual server is PUBLIC, and it appears in the virtual server list in ArcIMS Administrator when a new MapService is created.

Examples

1) This example extracts all active layers from the SanFranciscoExtract MapService. The SanFranciscoExtract MapService is based on the sf.axl file from the samples directory. Note that although it is a separate MapService type an Extract Service can use the same .axl file as an Image Service. However, in that case outputurl and outputpath must both be specified in the request.

```
<cf_arcims  action="extract"
   ServiceName="SanFranciscoExtract"
   ServerName="bear"
   ServerPort="5300"
   envelope="-122.54,37.65,-122.32,37.84"
   layerlistorder="false"
   outputpath="c:\ArcIMS\output"
   outputurl="http://bear/output" />

<cfoutput>
File: #OUT_FILEURL#
</cfoutput>
```

Variable OUT_FILEURL contains the URL of the .zip file, for example, "http://bear/output/bear3482264.zip".

2) In this example, only the County layer (ID 0) is extracted.

```
<cf_arcims  action="extract"
   ServiceName="SanFranciscoExtract"
   ServerName="bear"
   ServerPort="5300"
   envelope="-122.54,37.65,-122.32,37.84"
   layerlistorder="true"
   layervisiblelist="0"
   outputpath="c:\ArcIMS\output"
   outputurl="http://bear/output"
   generatehtml="true"/>
```

3) Using the CF_ARCIMS_LAYER subtag to specify the layers to extract:

```
<cf_arcims  action="extract"
   ServiceName="SanFranciscoExtract"
   ServerName="bear"
   ServerPort="5300"
   envelope="-122.54,37.65,-122.32,37.84"
   layerlistorder="true"
   outputpath="c:\ArcIMS\output"
   outputurl="http://bear/output"
   generatehtml="true">

   <cf_arcims_layer LayerID="0" Visible="true" />

</cf_arcims>
```

4) How to retrieve an extract file using the CFFTP ColdFusion tag:

```
<cf_arcims  action="extract"
   ServiceName="SanFranciscoExtract"
   ServerName="bear"
   ServerPort="5300"
   envelope="-122.54,37.65,-122.32,37.84"
   layerlistorder="false"
   outputpath="c:\ArcIMS\output"
   outputurl="http://bear/output"
   generatehtml="true" />

<cfoutput>
<cfftp action="GETFILE"
   server="bear"
   username="user_name"
   password="user_password"
   stoponerror="Yes"
   localfile="c:\temp\extract.zip"
   remotefile="#OUT_FILEPATH#"
   transfermode="BINARY"
   failifexists="Yes">

</cfoutput>
```

REQUEST

Purpose

Sends any ArcXML request from file or text input to ArcIMS Spatial Server.

Syntax

ArcXML request can be specified in AXLTEXT or AXLFILE attributes or by placing ArcXML tags inside the CF_ARCIMS tag.

Basic syntax

```
<CF_ARCIMS ACTION=”REQUEST”
   SERVICENAME=”service name”
   SERVERNAME=”server name”
   SERVERPORT=”server port”
   CUSTOMSERVICE=”Query/GeoCode”
   AXLFILE=”ArcXML file”
   AXLTEXT=”ArcXML text”
   ParseResponseAXL=”true|false”
   GENERATEHTML=”TRUE|FALSE” >
```

Extended syntax

Any ArcXML tag or sequence of tags can be placed inside this request, and it will be sent to the Spatial Server.

Attributes

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
AXLFILE	N	String	N/A	N/A	Name of the file containing an ArcXML to send to the Spatial Server with full path.
AXLTEXT	N	String	N/A	N/A	ArcXML string to send to the Spatial Server.
CUSTOMSERVICE	Y	Specified Values	N/A	Query, Geocode, “”	If using a GET_FEATURES ArcXML use Query. If using a GET_GEOCODE ArcXML request, use Geocode. For all other requests use an empty string (“”).
GENERATEHTML	N	Boolean	False	True, False	If true, the HTML table is generated in place of CF_ARCIMS tag during run time; if false, all variables are set but no table is generated.

Attribute Name	Required	Value Type	Default Value	Possible Values	Usage
PARSERESPONSEAXL	N	Boolean	False	True, False	If true, connector tries to parse the Spatial Server response. If successfully parsed, the response is put into output variables and queries similar to those for higher-level requests. If request is, for example, GET_IMAGE, response is parsed as for a GENERATEIMAGE request with all output variables for that request.
SERVERNAME	Y	String	N/A	N/A	Name of the server running ArcIMS Application Server.
SERVERPORT	Y	Integer	N/A	N/A	Server port number.
SERVICENAME	Y	String	N/A	N/A	Name of the MapService to be drawn.

Subtags

Any ArcXML code can be placed inside the CF_ARCIMS REQUEST request, and it will be sent to the Spatial Server. That ArcXML code replaces the AXLFILE or AXLTEXT attribute.

Output Variables

Variable Syntax	Always Generated	Variable Type	Usage
IN_REQUESTAXL	Y	String	ArcXML request sent to the Spatial Server. Used primarily for debugging.
OUT_RESPONSEAXL	Y	String	ArcXML request of the response received from the Spatial Server. Since the CF_ARCIMS tag does the ArcXML parsing internally and placesall relevant information in CF variables, this is used primarily for debugging.
OUT_ERROR	Y	String	Where the Spatial Server returns error messages.

- If PARSERESPONSEAXL=”True”, other output variables are generated depending on the request type.
- For an ArcXML GET_IMAGE request, variables are created as for a CF_ARCIMS GENERATEMAP request.
- For an ArcXML GET_FEATURES request, variables are created as for a CF_ARCIMS QUERY request.

- For an ArcXMLGET_SERVICE_INFO request, variables are created as for a CF_ARCIMS GETSERVICEINFO request.
- For an ArcXML GET_GEOCODE request, variables are created as for a CF_ARCIMS GEOCODE request.
- For an ArcXML GET_EXTRACT request, variables are created as for a CF_ARCIMS EXTRACT request.

Restrictions

All XML special characters (such as '<' and '>') must be replaced with the appropriate &; sequences if an ArcXML request is in the variables. For a list of special characters and their replacements, see the ColdFusion documentation.

Notes

- REQUEST is a general-purpose, low-level request. It is used to send any kind of ArcXML request to the Spatial Server, and it is often used to implement ArcIMS functionality not exposed through other CF_ARCIMS request types such as the dynamic renderer definition for a GET_IMAGE request. The full ArcXML must be supplied by the client, and it is not checked before it is sent to the Spatial Server so there is a higher possibility of error.
- The ArcXML request can be in the file specified by the AXLFILE attribute, in the string specified with the AXLTEXT attribute, or inside the CF_ARCIMS tag. Only one of these methods should be used.
- AXLTEXT can be enclosed in single quotes if the ArcXML text itself contains double quotes. However, if the ArcXML text contains both single and double quotes (for example, if it contains a QUERY tag with an SQL where statement) it cannot be specified directly in the AXLTEXT attribute. It either has to be composed as a variable or specified as a subtag.
- ArcXML is case sensitive. Tags are uppercase and attributes are lowercase.

Examples

1) Sending a simple GET_SERVICE_INFO request. It is similar to a GETSERVICEINFO CF_ARCIMS request:

```
<cfset axlRequest =
   '<ARCXML version="1.0"> <REQUEST> ' &
   '<GET_SERVICE_INFO fields="true" envelope="true" renderer="false" /> ' &
   '</REQUEST> </ARCXML> ' />

<cf_arcims  action="request"
   ServiceName="SanFrancisco"
   ServerName="bear"
   ServerPort="5300"
   CustomService=""
   axltext= "#axlRequest#"
   generateHTML="true"
   ParseResponseAXL="true" />
```

2) This example defines a temporary layer based on a census blockgroup layer with a query that selects areas with median age less than 35 and creates a renderer that highlights those areas in green. This kind of query cannot be made dynamically with other CF_ARCIMS tags. This example also demonstrates the use of ArcXML code inside a CF_ARCIMS tag.

```
<cf_arcims  action="request"
   ServiceName="Demog"
   ServerName="bear"
   ServerPort="5300"
   CustomService=""
   generateHTML="true"
   ParseResponseAXL="true">

   <ARCXML version="1.0.1">
      <REQUEST>
        <GET_IMAGE>
          <PROPERTIES>
                <IMAGESIZE width="400" height="400"/>
                <ENVELOPE minx="-122.54" miny="37.65" maxx="-122.32" maxy="37.84"/>
                <LAYERLIST order="false" >
                        <LAYERDEF id="3" visible="true"/>
                        <LAYERDEF name="Temp_Layer" visible="true"/>
                </LAYERLIST>
          </PROPERTIES>
          <LAYER type="FeatureClass" name="Temp_Layer" visible="true">
                <DATASET fromlayer="Census Blockgroups"/>
                <QUERY where="MED_AGE &lt; 35"/>
                <SIMPLERENDERER>
                        <SIMPLEPOLYGONSYMBOL fillcolor="150,255,150" filltype="solid"
                        boundarywidth="2" boundarycolor="0,255,0"S />
                </SIMPLERENDERER>
         </LAYER>
       </GET_IMAGE>
    </REQUEST>
   </ARCXML>

</CF_ARCIMS>
```

Application development and debugging tips

Developing mapping and GIS Web applications using ColdFusion and ArcIMS can be as basic as placing a on the page or performing more advanced queries and presenting the results.

Using the REQUEST action in place of other actions

Some of the actions defined for the CF_ARCIMS tag do not expose the full functionality of ArcIMS. Fortunately, you can use the REQUEST action in cases where higher-level actions do not expose particular functionality. The REQUEST action takes an ArcXML request as an argument, and sends it to theArcIMS Spatial Server. The use of REQUEST assumes knowledge of ArcXML syntax. See the *ArcXML Programmer's Reference Guide* for a complete reference to ArcXML.

REQUEST in place of GENERATEMAP

You can typically create maps using the GENERATEMAP action. That is true for maps that contain acetate layer objects of the same type and symbology, or highlighted features with unique keys. However, if an acetate layer contains a number of differently symbolized objects, or several sets of features that are highlighted differently or, if those features do not have a unique key, the developer needs to use REQUEST instead of GENERATEMAP.

REQUEST in place of QUERY

You use the QUERY action to make a spatial, attribute, or a combined query. However, it is not possible to use join tables with non-SDE data sources. Also, if buffer is used, the features selected by the buffer are returned, but the buffer geometry itself cannot be returned. If you want to show the buffer on the map image, you must use the REQUEST action.

REQUEST in place of LEGEND

When your application uses layers set with scale factors for drawing, you might want to use the REQUEST action to generate the legend. The LEGEND action create the legend with all layers in the MapService, regardless if they are turned off due to scale factors. It makes sense to combine a GET_IMAGE and LEGEND request in one REQUEST tag, since it creates the map and the legend using only one ArcXML request.

REQUEST in place of EXTRACT

The EXTRACT action is used to give a user a quick and easy way to get the features currently on the map into a shapefile. EXTRACT is based on GET_IMAGE syntax and not on GET_FEATURES. However, some situations require features to be extracted based on an attribute query or spatial query more complicated than envelope search. In this case, the EXTRACT request can't be used because it doesn't allow for any kind of query or external table joins. This is only possible using an ArcXML request from the REQUEST action.

ColdFusion Connector tags and ArcXML requests

The ArcIMS ColdFusion Connector translates ColdFusion custom tag requests into ArcXML, sends the ArcXML request to the Spatial Server, takes the response from the Spatial Server, parses it, and puts the result into ColdFusion variables. Every CF_ARCIMS action, except REQUEST, creates one ArcXML request type.

CF_ARCIMS Action	ArcXML Tag
GENERATEMAP	<GET_MAP>
QUERY	<GET_FEATURES>
GEOCODE	<GET_GEOCODE>
EXTRACT	<GET_EXTRACT>
IDENTIFY	<GET_FEATURES>
GETMAPSERVICES	<GET_SERVICES>
GETSERVICEINFO	<GET_SERVICEINFO>

Request and Response

It is often useful to review the ArcXML request sent to the Spatial Server by the connector and the ArcXML response returned by the Spatial Server. Those ArcXML strings are stored in IN_REQUESTAXL and OUT_RESPONSEAXL ColdFusion variables.

For example, the code below creates an ArcXML request that is shown following.

```
<CF_ARCIMS ACTION=”GENERATEMAP”
   SERVERNAME=”bear”
   SERVERPORT=”5300"
   SERVICENAME=”SanFrancisco”
   ENVELOPE=”-122.46,37.76,-122.42,37.79"
   IMAGESIZE=”400,400"
   LAYERLISTORDER=”true”
   GENERATEHTML=”true”>

   <CF_ARCIMS_LAYER  LayerID=”2"   Visible=”true” >
   <CF_ARCIMS_LAYER  LayerID=”3"   Visible=”true” >
   <CF_ARCIMS_LAYER  LayerID=”5"   Visible=”true” >

   </CF_ARCIMS>

<cfoutput>
```

```
<?xml version=”1.0"?>
 <ARCXML version=”1.0.1">
 <REQUEST>
 <GET_IMAGE>
<PROPERTIES>
<IMAGESIZE width=”400" height=”400"/>
<ENVELOPE minx=”-122.46" miny=”37.76" maxx=”-122.42" maxy=”37.79"/>
<LAYERLIST order=”true” >
<LAYERDEF id=”2" visible=”true”/>
<LAYERDEF id=”3" visible=”true”/>
<LAYERDEF id=”5" visible=”true”/>
</LAYERLIST>
</PROPERTIES>
</GET_IMAGE>
 </REQUEST>
 </ARCXML>
```

This request is in the IN_REQUESTAXL variable. The Spatial Server returns a response.

```
<?xml version=”1.0"?>
<ARCXML version=”1.0">
<RESPONSE>
<IMAGE>
<ENVELOPE minx=”-122.46000000" miny=”37.75500000" maxx=”-122.42000000"
maxy=”37.79500000" />
<OUTPUT file=”C:\ArcIMS\output\SanFrancisco_BEAR3273501.png” url=”http://bear/
output/SanFrancisco_BEAR3273501.png” />
</IMAGE>
</RESPONSE>
</ARCXML>
```

The ColdFusion Connector parses the response and puts ENVELOPE and OUTPUT tag values in appropriate ColdFusion variables. Entire response is in OUT_RESPONSEAXL.

It is not recommended to try to show content of IN_REQUESTAXL and OUT_RESPONSEAXL on the result page, even for debugging. The browser will probably try to interpret it as XML code and return an error. Instead, put the variable values inside an HTML comment (not the ColdFusion comment), so the browser will not try to interpret them.

The code that deals with IN_REQUESTAXL and OUT_RESPONSEAXL follows.

```
<cfoutput>
<!—
Request:
#IN_REQUESTAXL#

Response:
#OUT_RESPONSEAXL#
—>
</cfoutput>
```

Server log files

If the connector tag returns an error, IN_REQUESTAXL and OUT_RESPONSEAXL will not be defined. To find the problem in that case, the developer should check the server log file. Server log files are placed at %AIMSHOME%\server\log for Windows NT and $AIMSHOME/log for UNIX. The file name is composed of VirtualServerType_HostName_Number.log, so the log file for an image server on host bear may be ImageServer_BEAR_327.log.

The log file contains the full code of the ArcXML request, processing information, errors (if any), and the full code of the ArcXML response sent to the client. Processing information depends on the Server type. For Image Server, for example, for every layer there is the time spent searching and retrieving layer name features for that layer and the number of features processed. That information can be very useful for tuning.

For example, for the previous request, the log file has the following entry:

```
[Fri May 05 12:59:28 2000][327 350 INFO1] Begin Request
[Fri May 05 12:59:28 2000][327 350 INFO3] REQUEST:
<GET_IMAGE>
<PROPERTIES>
<ENVELOPE minx=”-122.46" miny=”37.76" maxx=”-122.42" maxy=”37.79" />
<IMAGESIZE width=”400" height=”400" />
<LAYERLIST order=”true” >
<LAYERDEF id=”2" visible=”true”/>
<LAYERDEF id=”3" visible=”true”/>
<LAYERDEF id=”5" visible=”true”/>
</LAYERLIST>
</PROPERTIES>
</GET_IMAGE>

(continued on the next page)
```

```
[Fri May 05 12:59:28 2000][327 350 INFO1] SERVICE: SanFrancisco
[Fri May 05 12:59:28 2000][327 350 INFO2] ArcXML Parse Time: 0.360000s
[Fri May 05 12:59:28 2000][327 350 INFO2] RENDERER SETUP: 0.030000s
[Fri May 05 12:59:28 2000][327 350 INFO2] FEATURE LAYER: Highways
[Fri May 05 12:59:28 2000][327 350 INFO2] DATA SEARCH TIME: 0.020000s
[Fri May 05 12:59:29 2000][327 350 INFO2] GR FEATURES PROCESSED: 213
[Fri May 05 12:59:29 2000][327 350 INFO2] DATA RETRIEVAL TIME: 0.240000s
[Fri May 05 12:59:29 2000][327 350 INFO2] FEATURE LAYER: Agencies
[Fri May 05 12:59:29 2000][327 350 INFO2] DATA SEARCH TIME: 0.000000s
[Fri May 05 12:59:29 2000][327 350 INFO2] GR FEATURES PROCESSED: 0
[Fri May 05 12:59:29 2000][327 350 INFO2] DATA RETRIEVAL TIME: 0.000000s
[Fri May 05 12:59:29 2000][327 350 INFO2] FEATURE LAYER: Theaters
[Fri May 05 12:59:29 2000][327 350 INFO2] DATA SEARCH TIME: 0.020000s
[Fri May 05 12:59:29 2000][327 350 INFO2] GR FEATURES PROCESSED: 10
[Fri May 05 12:59:29 2000][327 350 INFO2] DATA RETRIEVAL TIME: 0.000000s
[Fri May 05 12:59:29 2000][327 350 INFO2] TOTAL PROCESSING TIME: 0.410000s
[Fri May 05 12:59:29 2000][327 350 INFO2] OUTPUT TIME: 0.331000s
[Fri May 05 12:59:29 2000][327 350 INFO3] RESPONSE:
<?xml version=”1.0"?>
<ARCXML version=”1.0">
<RESPONSE>
<IMAGE>
<ENVELOPE minx=”-122.46000000" miny=”37.75500000" maxx=”-122.42000000"
maxy=”37.79500000" />
<OUTPUT file=”C:\ArcIMS\output\SanFrancisco_BEAR3273501.png” url=”http://bear/
output/SanFrancisco_BEAR3273501.png” />
</IMAGE>
</RESPONSE>
</ARCXML>
[Fri May 05 12:59:29 2000][327 350 INFO2] Total Request Time: 1.152000s
[Fri May 05 12:59:29 2000][327 350 INFO1] End Request
```

The total processing time was 1.15 seconds and there are 213 features processed from the Highways layer and 10 from the Theaters layer.

Comparison of CF_ARCIMS and CFX_ESRIMAP

There are two types of ArcIMS ColdFusion connector tags—CF_ARCIMS and CFX_ESRIMAP. CF_ARCIMS tags and the rest of the CF_* tags described in this document are wrapper custom tags written as .cfm files in the ColdFusion custom tag directory. They wrap the functionality of the CFX_ESRIMAP tag. CFX_ESRIMAP is a basic connector tag written in C++ using the ColdFusion API.

CFX_ESRIMAP is very similar to CF_ARCIMS. It supports all of the same attributes and action types. The only difference is that CFX_ESRIMAP does not support any subtags. The subtags often handle the escaping of string attributes, so you may need to implement the escaping of the strings. All data for the query must be supplied by the attributes.

If you encounter a problem in an ArcXML request sent by the ColdFusion Connector, you can try to replace CF_ARCIMS with CFX_ESRIMAP and test the code again.

The example below show the equivalent CF_ARCIMS and CF_ESRIMAP code for a query. The first uses the CF_ARCIMS_SQL subtag, the latter does not.

CF_ARCIMS

```
<cf_arcims  action=”query”
ServiceName=”SanFrancisco”
ServerName=”bear”
ServerPort=”5300"
LayerID=”2"
layerdisplayfields=”##ALL##”
featurelimit=”100"
   generatehtml=”false”>
   <CF_ARCIMS_SQL>
      HWYNAME=’MARKET ST’
   </CF_ARCIMS_SQL>
</cf_arcims>
```

CFX_ESRIMAP

```
<cfx_esrimap  action=”query”
ServiceName=”SanFrancisco”
ServerName=”bear”
ServerPort=”5300"
LayerID=”2"
layerdisplayfields=”##ALL##”
featurelimit=”100"
generatehtml=”false”
where=”HWYNAME=’MARKET ST’”/>
```